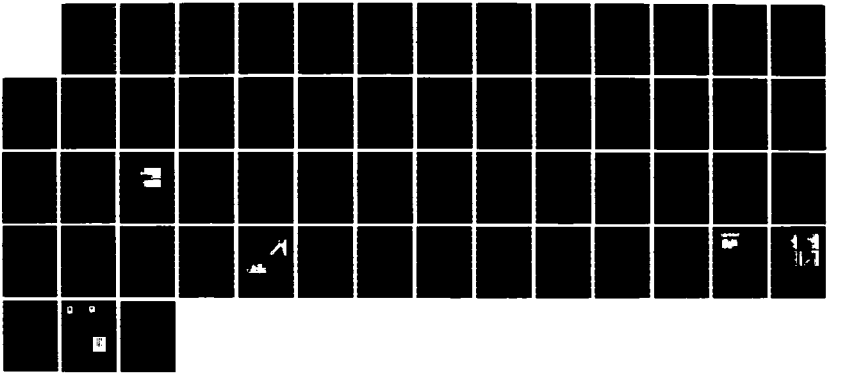
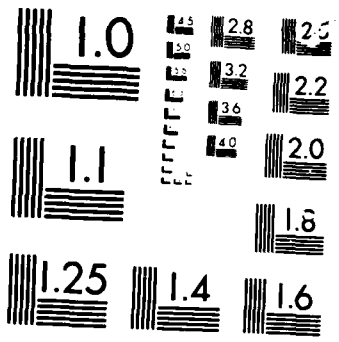
AD-A169 068 A VLSI-BASED HIGH-PERFORMANCE RASTER IMAGE SYSTEM(U) 1/1
NORTH CAROLINA UNIV AT CHAPEL HILL DEPT OF COMPUTER
SCIENCE H FUCHS ET AL. 08 MAY 86 ARO-21107.3-EL-A
UNCLASSIFIED DAAG29-83-K-0148 F/G 9/5 NL





A VLSI-based High-performance Raster Image System

Final Report

Henry Fuchs
John Poulton

May 4, 1986

U. S. Army Research Office

DARPA Contract DAAG 29-83-K-0148

University of North Carolina
Department of Computer Science
Chapel Hill, NC 27514

Our research objective is to create affordable, high-performance, 3-D raster display systems that harness the highly parallel computational power of custom VLSI circuits. We want to make this power available in an open system that encourages invention of new parallel algorithms for image generation and image processing. The system has many potential applications including flight and tactical displays, computer-aided mechanical design, and medical diagnosis and therapy.

As in other systems, ours includes an array of memory chips forming an image buffer from which the video screen is refreshed. The novel feature of our design is that our memory chips have been custom-designed to include processing circuitry so that each pixel (dot on display) can carry out its own image-generation calculations. Calculations are distributed throughout each chip in such a way that only a very small amount of circuitry is required for each pixel. In our current chips, this processing circuitry takes up about one-third of the total area, with the remainder devoted to pixel memory.

During the three years of this work, we have achieved significant experimental results. We have developed algorithms for fast image generation, including very rapid rendering of spheres and shadow-casting (our system is the only graphics machine, to our knowledge, that can render true shadows in real time). We have also designed two custom integrated circuits for the machine; the new memory chip is the fastest memory so far designed in the university/VLSI community. These chips have been built into a running prototype. A full-scale, full-speed machine (512x512 pixels, 30,000 smooth-shaded triangles per second) is nearing completion.

We next plan to explore two new architectural innovations: one will render curved surfaces directly and the other will yield a 5-fold improvement in speed. We also plan to implement our custom chips in newly-available 1.2μ CMOS fabrication, to make our system practical for personal workstations. We plan to build two complete, full-scale machines, one of which will be delivered to our collaborators at the Army Ballistic Research Laboratory. These systems will become the platforms for developing new algorithms and applications for [illegible] modeling and for image enhancements, such as textures, shadows, and anti-aliasing.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: ARO 21107.3-EL-A	2. GOVT ACCESSION NO.: AD-A169068
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE (and Subtitle): A Prototype for a VLSI-based High-performance Raster Image System	5. TYPE OF REPORT & PERIOD COVERED: Final Report, 19 Sep 83 - 18 Mar 86
	6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): Henry Fuchs, John Poulton	8. CONTRACT OR GRANT NUMBER(s): DARPA Contract DAAG 29-83-K-0148
9. PERFORMING ORGANIZATION NAME AND ADDRESS: University of North Carolina, Department of Computer Science, New West Hall 035A Chapel Hill NC 27514	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11. CONTROLLING OFFICE NAME AND ADDRESS: U. S. Army Research Office, Post Office Box 12211, Research Triangle Park, NC 27709	12. REPORT DATE: May 8, 1986
	13. NUMBER OF PAGES: 52
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):	15. SECURITY CLASS. (of this report): unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

THE VIEW, OPINIONS, AND/OR FINDINGS CONTAINED IN THIS REPORT ARE THOSE OF THE AUTHOR(S) AND SHOULD NOT BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION, POLICY, OR DECISION, UNLESS SO DESIGNATED BY OTHER DOCUMENTATION.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

VLSI, Raster Graphics, Processor-enhanced Memories

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Note: This work also sponsored by
National Science Foundation
Grant ECS-8300970

Table of Contents

1. Introduction

Research Objective

To create high-performance 3-D raster display systems that harness the highly parallel computational power of custom VLSI circuits and to make this power available in an open, programmable system that encourages invention of new parallel algorithms for image generation and image processing.

Background

Desk-top workstations cannot today be equipped with high-performance 3-D graphics systems mainly because of the lack of fast, affordable hardware to handle the very demanding screen-oriented tasks of image rendering (*e.g.*, smooth shading, visibility determination, shadow casting). The Pixel-planes design attacks this problem at the raster image memory (the 'frame buffer' in a conventional graphics system). Our system is based on custom-designed, logic-enhanced VLSI memory chips in which image memory elements are combined with a small amount of processing circuitry that carries out the most burdensome calculations in image rendering. These calculations are done in parallel for all pixels on the display by a binary tree of tiny processing elements.

Pixel-planes is an open, programmable architecture rather than a black box that provides a single, hard-wired solution. It encourages users to explore new parallel algorithms for image display and processing. Several such algorithms have been developed from the architectural description of the design, with only the promise of a real machine.

Among the many possible applications for Pixel-planes are solid modeling systems for computer-aided mechanical design and analysis, diplays for medical diagnosis and therapy, and molecular modeling systems.

The project has been under way for four years and has produced considerable experimental results, including a variety of graphics algorithms and three generations of small prototype displays. We are currently scaling up the latest full-speed prototype to a full-size display system (512 x 512 pixels, 72 bits per pixel) to be completed by summer, 1986.

2. Review of Pixel-planes Project

2.1 System Overview

This section presents a brief overview of the Pixel-planes system. Figure 1 shows a conceptual block diagram of a conventional interactive 3-D graphics system and Pixel-planes' relationship to it. In such a system, a user interacts with a 3-D scene stored in a database, typically consisting of a list of vertices of polygons that tile the surfaces of objects in the scene.

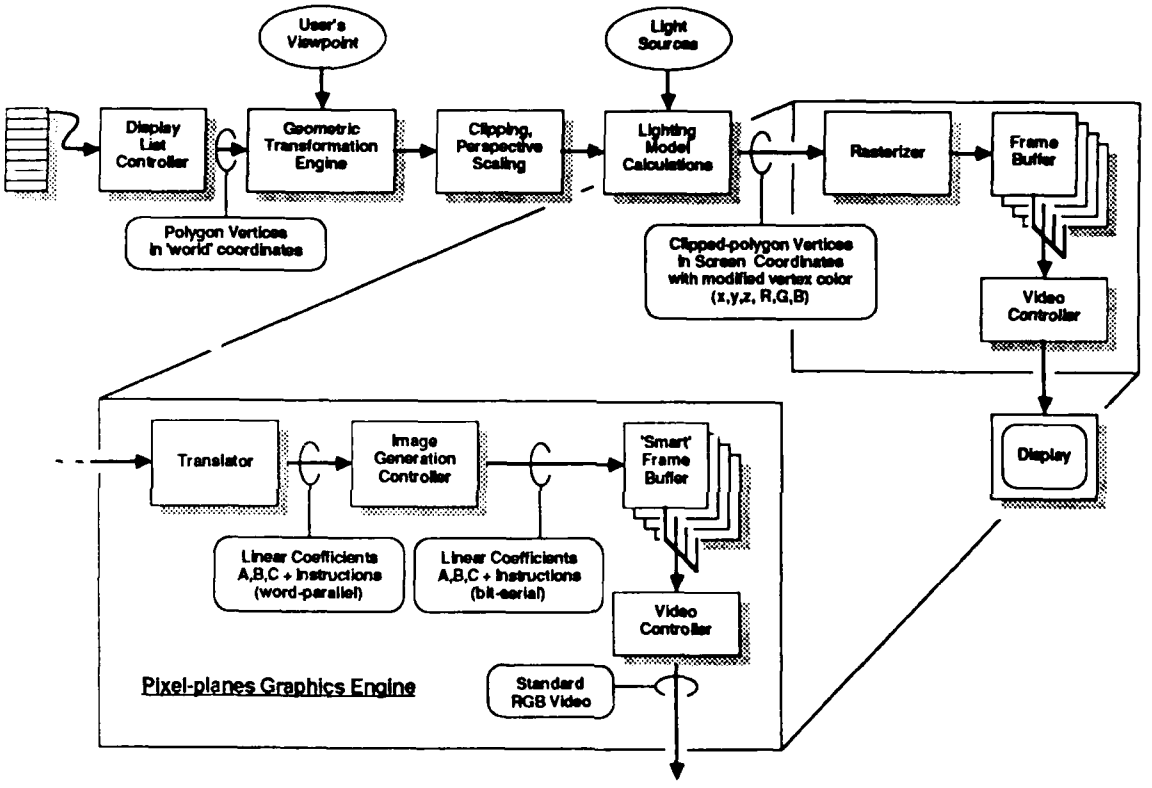


Figure 1: Relationship between Pixel-planes Graphics Engine and a conventional system for rendering 3-D images. The Engine replaces rasterizer and frame buffer with a 'smart' frame buffer built from custom, logic-enhanced memory chips.

Real-time interaction depends on computation in a processing pipeline whose elements perform the following tasks:

1) **Transform** the scene according to the user's position and orientation; this step produces a list of polygon vertices in 'eye' coordinates.

2) **Clip** away parts of objects that are outside the field of view.

3) **Color each polygon vertex** according to a lighting model that takes into account the position, intensity, and color of user-specified light sources and the color and orientation of the polygon.

4) Scale for perspective by drawing closer together objects that are farther away; the result is a list of polygon vertices whose coordinates are given in 'screen' coordinates.

5) Scan-convert each polygon, determining which pixels are inside that polygon.

6) Remove hidden surfaces by determining which pixels in the current polygon are obscured by previously processed polygons.

7) Color each pixel, interpolating between vertex colors.

8) Store pixels in a frame buffer, from which the system can:

9) Refresh the video display.

Steps (5)-(7) clearly represent the performance bottleneck in current graphics systems. Many current, affordable graphics systems can handle polygon transformations, steps (1)-(4), sufficiently rapidly to support real-time interaction with complex wire-frame images or with a restricted class of flat-shaded polygonal images. There exists, however, no affordable solution to the problem of painting highly realistic, fully rendered 3-D scenes, perhaps with image enhancements such as smooth shading, anti-aliasing (to remove pixel artifacts), shadows, textures, transparent surfaces, fog effects, and so forth.

The Pixel-planes design attacks this problem by replacing the conventional rasterizer/frame buffer with a 'smart' frame buffer that not only *stores* a digital image but also performs much of the calculation needed to *generate* the image. This 'smart' frame buffer is built from custom chips in which conventional memory circuits are combined with some processing circuitry that allows computations to be carried out in parallel for all pixels in the display.

The processing circuitry in the enhanced memory chips can perform two kinds of operations:

- Evaluate linear expressions of the form $F(x,y) = Ax + By + C$ simultaneously for all pixel locations (x,y). A,B,C are data broadcast to the memory chips.
- Perform pixel-local arithmetic and logical operations on data stored at the pixel and on the linear expressions.

The strategy for applying this system to a given graphics image generation problem is to re-cast the problem into a form that requires only linear-expression evaluation and pixel-local operations. To make use of these operations, the Translator (see Figure 1) converts the conventional description of graphics primitives (*e.g.*, a list of vertices for each polygon) into the form of coefficients A,B,C and instructions for the pixel-local processors. The Image Generation Controller converts these coefficients and instructions into bit-serial form and broadcasts them to the array of enhanced memory chips.

2.2 Algorithms

A basic set of graphics operations for rendering convex polygons includes:

Scan-conversion. At the beginning of each polygon, all pixels are enabled. For each edge of a polygon, the Translator calculates the coefficients for an expression $F(x,y)=Ax+By+C$, where $F(x,y)=0$ defines the points that lie along a line connecting two adjacent vertices, and broadcasts A,B,C and an 'edge' instruction to the array of enhanced memory chips. Each pixel-processor examines the sign of its value for the expression F. If negative, the pixel is outside the current

polygon and is disabled for further processing. If positive, the pixel remains enabled. As each succeeding edge is processed, pixels are disabled by half-planes, until, after all edges are processed, only those inside the polygon remain enabled.

Hidden-surface Elimination. In an implementation of the standard depth-buffer (z-buffer) algorithm, the Translator calculates a set of coefficients for the expression z=F(x,y)=Ax+By+C, the planar equation for the current polygon's surface, and broadcasts A,B,C and z-compare instructions to the memory chips. Each pixel maintains the z-coordinate of the closest visible polygon so far processed. Each pixel compares the value of the linear expression with its stored z; if the new z is farther away than the stored z, the pixel is disabled (the new polygon is obscured at that pixel); if closer, the pixel remains enabled.

Smooth Shading. The Translator calculates, for each color, coefficients for the expression Intensity=Ax+By+C and broadcasts these coefficients and 'color' instructions to the memory chips. A multi-sided polygon is at this stage broken into triangular patches for shading. For each patch, and for each primary color, enabled pixels update their color-intensity buffers with the new color intensity.

A number of other algorithms have been developed, including casting shadows, drawing and shading spheres, painting textures, and anti-aliasing edges. They are described in Appendix 1.

2.3 Chip Design

Most of the system's processing power derives from Pixel-planes' novel Linear Expression Evaluator (LEE), a binary-tree structure distributed uniformly over an array of identical enhanced memory chips that form the 'smart' frame buffer; its principle is illustrated in Figure 2.

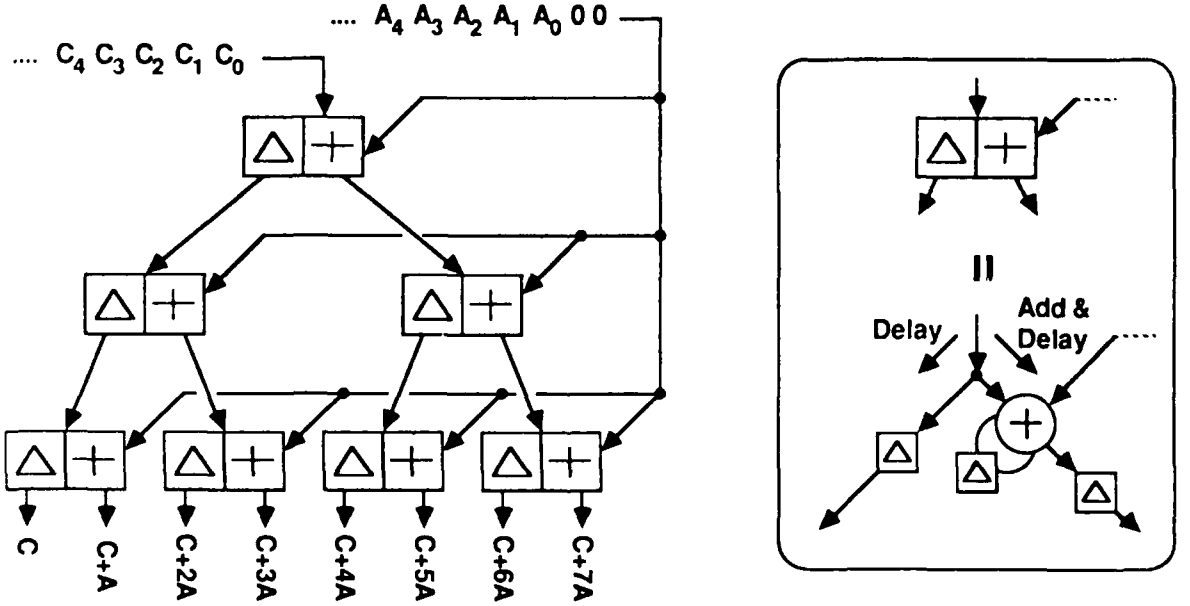


Figure 2: Fundamental operation of the Linear Expression Evaluator.

Each node of the tree takes in a serial bit-stream at its 'top' input, passing it unchanged, but delayed by one clock cycle, to the left-hand output. The right-hand output is formed by adding a second bit-stream from the 'side' input; bit-serial addition injects a one-cycle delay in the right branch and requires the usual local carry register shown in the figure. In effect, each tree node forms both possible values of a partial product contributing to Ax + C. Because of the leading 0's in the A bit stream, A_0 arrives at the bottom level of the tree at the same time as C_0; thus, 1*A is added at each node at the bottom level. At the second level, A_0 is bit-aligned with C_1, so that 2*A is added at this level, and so on. In general an n-level tree can generate 2^n distinct values for an expression of the form Ax+C.

For a system with 1024x1024 pixels, a 10-level X-tree is required to generate the 1024 distinct values of Ax+C. The outputs of this tree in turn feeds the 'top' input of 1024 Y-trees, and this ensemble generates the 2^{20} distinct values of Ax+By+C. Note that the cost in silicon area for this entire structure is only a single tree node (three one-bit registers and one bit-serial adder) for each pixel, but the performance is equivalent to a full 20-stage serial multiplier at every pixel.

A simplified block diagram of the Pixel-planes memory chip is shown in Figure 3.

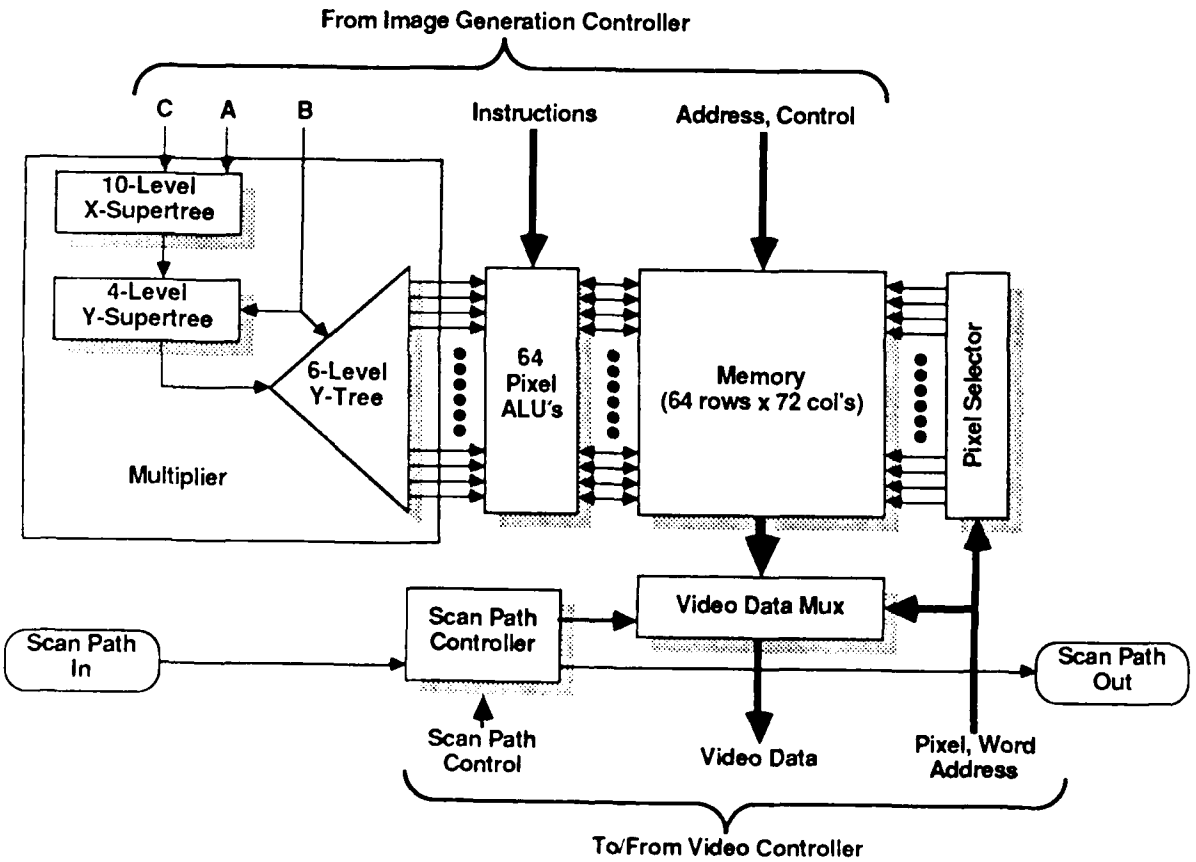


Figure 3: Block diagram of Pixel-planes enhanced memory chip. Pxpl4.0 chips contain one such module (64 pixels x 72 bits); Pxpl4.1 chips contain two modules (128 pixels x 72 bits).

In addition to the Linear Expression Evaluator, these chips contain an array of tiny bit-serial ALU's that perform pixel-local operations (one ALU for each pixel), the pixel memory itself, and circuitry for scanning out data to refresh a display.

Only a small subset of the pixels in a display can be placed on a single chip, so the LEE is distributed over many chips. This is done by building a subtree that covers only the pixels on the chip (organized as a vertical column); the remainder of the tree is implemented by circuitry that maps a path through the complete tree from the root of the complete tree to the root of the subtree. Called a 'supertree', this path is programmed during system initialization, loading each chip with its pixel-column address in the display. The supertree construct considerably simplifies the system-level implementation of our system at little cost (about 5%) in silicon area. In our current chips, about 18% of the circuit area is devoted to the LEE, about 12% to the ALU array, and the remaining 70% to memory.

3. Accomplishments

Our project has achieved considerable results during the past two years; these are described in detail in Appendix 1 (Hardware) and Appendix 2 (Algorithms). We here describe the highlights of these accomplishments.

3.1 Custom Chips

Between November, 1984, and May, 1985, our team designed two custom chips that were fabricated, tested, and integrated into a working display system.

The first of these, the Coefficient Serializer, was relatively small (4.6x6.8 mm, 8,000 transistors, 4-μ nMOS), but it solved a particularly difficult system-level problem, one whose details took much time and simulation to work out. A standard-logic implementation would have required roughly 200 TTL packages, forcing a two-board implementation of the Image Generation Controller. The Coefficient Serializer chip had a particularly lucky history. Subsequent to finishing the layout, simulation of the extracted circuit ran perfectly after only a single minor fix. In April, 1985, we received 7 prototype chips from MOSIS and were surprised and delighted to find that all of them ran perfectly and at the design speed of 10MHz.

Pxpl 4.0, the new design of the custom memory chip, was a 34,000-transistor 4μ nMOS design in a large (7.9x9.2 mm) die. Memory size was 64 pixels by 72 bits/pixel; of the internal active circuit area, 70% was devoted to memory, 12% to the pixel-local ALU's, and 18% to the LEE. Prototype parts were received from MOSIS in mid-April and were found to work correctly and at the design speed. These parts were immediately incorporated into a 32-chip multi-board display able to support near real-time interaction with simple images.

During the summer of 1985, the design was recast in a 3μ part, Pxpl4.1, with two modules identical to the 4.0 design to give a total of 128 pixels. These parts also worked at speed on first fabrication, and, installed on a new custom printed-circuit card, were incorporated into the prototype display. At this point, we decided that the design was mature enough to incorporate into a full-scale, full-speed display. During the fall of 1985, we carried out a thorough design review of the 4.1 chip design to qualify it for quantity fabrication. The careful scrutiny of the review revealed one minor problem, corrected in Pxpl4.2, and this new version is now in quantity fabrication. Altogether, versions 4.1 and 4.2 have been fabricated by MOSIS five times with three different vendors; yield has been better than 20% on average, quite reasonable for such a large die.

3.2 System Building

In spring, 1985, we built our first system capable of expansion to usable size. This system verified our design for a multi-board memory system and for the two special controllers (Image Generation and Video). The system was housed in a Multibus card cage with memory chips mounted on custom PC boards. Each of the two controllers was built on a double-height (12"x12") Multibus wire-wrapped card, and each contained about 100 IC's; system integration and debugging required only about 2 weeks. The graphics pipeline and Translator for this prototype were implemented in software running in a UNIX workstation, with coefficients and instructions transmitted to the Image Generation Controller via programmed I/O. This arrangement was able to support near real-time interaction with very simple images through a keyboard interface. The system demonstrated algorithms for shadow casting and for rendering shaded spheres.

In fall, 1985, with the verification of the 4.1 version of the enhanced memory chip, we began designing a full-scale, full-speed system capable of displaying 512x512 pixels, 72 bits/pixel, and equipped with a hardware graphics pipeline. At the time of this writing, the hardware for this system is essentially complete and running; only quantity fabrication and testing of chips and boards remains to be done. During this time we have also constructed a high-speed VLSI tester capable of testing wafers as well as packaged parts.

Figure 4 is an overview of the current Pixel-planes system. A UNIX workstation host supervises configuring the system and loading graphics data bases into it. During operation, the host does nothing but service graphics input devices and generate transformation matrices to be sent to the graphics pipeline.

The central block contains the graphics pipeline and special controllers. At the moment we plan to use two, and possibly up to four, Mercury Systems 3232 floating-point array processors for the pipeline, connected by special hardware queues. In the two-processor version, the first AP controls the display list, does geometric transformations, perspective division, and lighting; the second is the Translator.

The Image Generation and Video controllers in this current system are minor modificaitons of the designs verified in the previous prototype. They send control and data in broadcast form to the array of memory chips in the frame buffer, shown in the bottom block. This is simply a physical structure to hold up to 2,048 custom memory chips, and to daisy-chain chips and boards together to collect the video output. A serial scan path links every chip in the system to the video controller. During setup, this path loads configuration registers on each chip; during operation, it carries a series of tokens that act as chip-selects for scan-out.

The system is housed in a 5-foot-high, 30"-wide rack donated to us, along with most of the system power supplies, by Data General Corporation. This enclosure also provides a well-designed air-handling system for removing heat from the frame buffer. Two custom PC board designs were used in the frame buffer. A large backplane carries control/data signals and a total of up to 900 amps of DC power to 32 daughter boards. The backplane design features extra-thick copper traces for current handling, provisions for heavy copper bus bars, and press-fit connectors for daughter boards. Each daughter board contains 70 locations for enhanced memory chips, packaged in 84-pin pin-grid arrays; the design is fairly aggressive with .008" lines and spaces and 8000 vias.

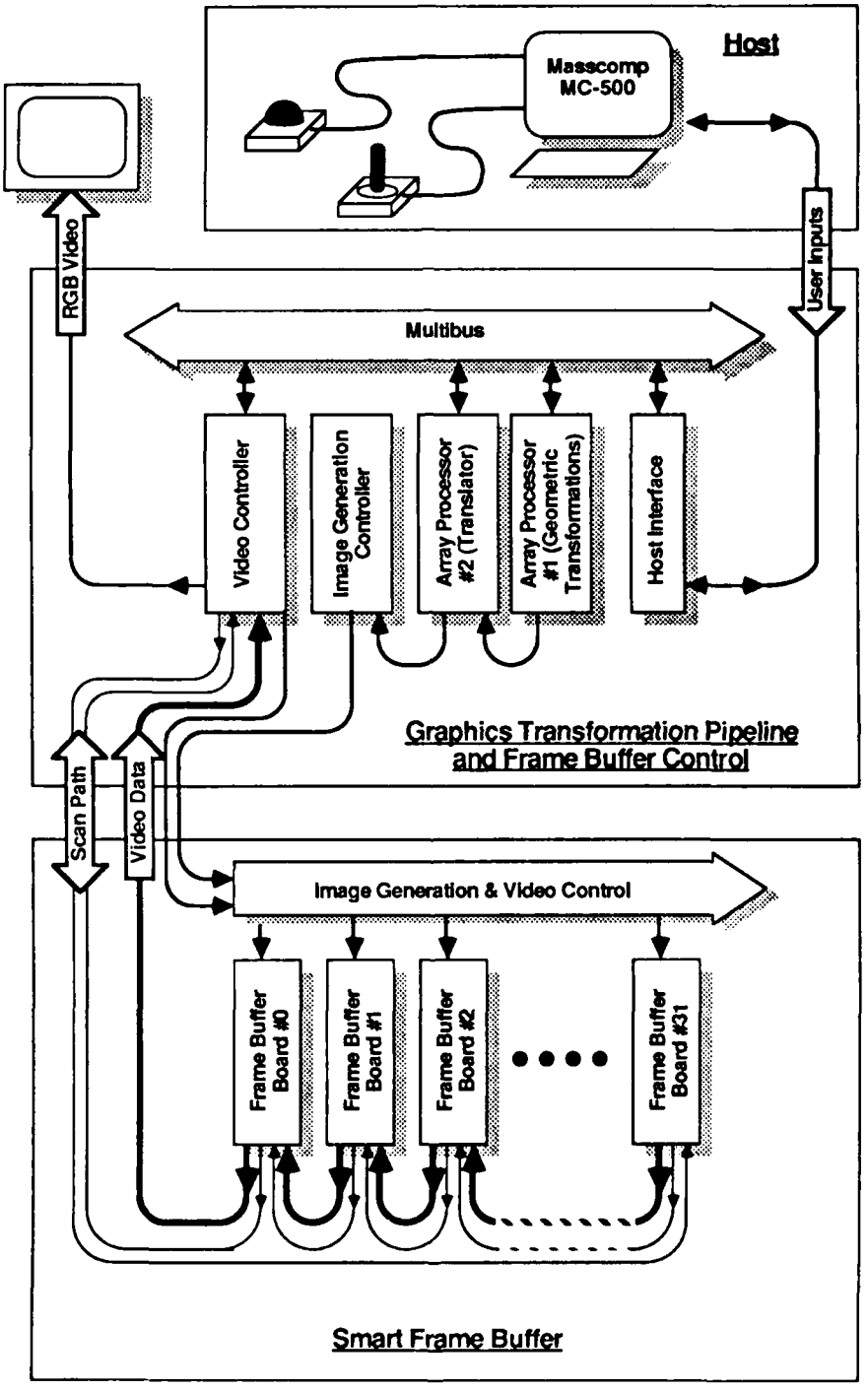


Figure 4: Overview of current Pixel-planes system.

In spite of the complexity of the current system, only about two weeks were required for testing and system integration, about the same time as for the previous prototype system. No serious problems were encountered during this phase of construction, and we take this as evidence of the success of our system design methodology. We took special care over several design issues:

- Detailed (gate-level) simulation to debug the logic design and 'architectural'-level simulation to test and refine algorithms.
- Careful design and specification of system interfaces, particularly at the chip I/O boundaries.
- Use of 'signal typing' borrowed from strongly-typed programming languages.
- Adoption of a simple and effective on-chip clocking strategy that greatly reduces the possibility of skew; this method is intrinsically linked to hot-clocking for nMOS implementations of our memory chip [Seitz, 1985].
- Complete, detailed, and continuously updated documentation.

3.3 Algorithms

A number of new algorithms have been developed for the architecture during the past year or so. Details are included in Appendix 2. Highlights of this work are:

Fast Spheres. A clever algorithm, originally suggested by Fred Brooks, allows circles and spheres to be rendered very rapidly. Essentially, one factors the implicit equation of a circle into a linear part and a quadratic part. The linear part has exactly the form Ax+By+C, and encodes the circle's center-position and radius. The quadratic part, $x^2 + y^2$, differs for every pixel, but is the same for all circles; it can therefore be pre-computed and loaded into each pixel during system initialization (this actually takes only about 100μsec in our current system). To draw a circle, each pixel simply adds its quadratic term to the incoming linear term to compute the circle equation. Rasterization proceeds just as though the system were drawing a polygon with one edge. We have demonstrated smooth shading and visible-surface calculations using variations of this technique; we intend to explore the potential of this algorithm in a real application--molecular modeling.

Shadows. Jeff Hultquist has developed a variation of Pixel-planes visible-surface algorithm that determines which pixels lie inside the shadow frustum of a light source and a polygon. Once identified, the color values of these pixels are modified to give the appearance of shadowed illumination. This shadow volume calculation is carried out as a post processing step, after the full image has been rendered, and it takes less time that rendering. To our knowledge, Pixel-planes is the only graphics system that can display shadows without treating them as separate objects.

Medical Image Enhancement. Adaptive histogram equalization (AHE) is a method of compressing the intensity range of computed tomographic (CT) images so that the view can perceive contrast and details. The method requires computation of a distribution of intensities at every pixel in the image, and so is far too time-consuming to be practical on uni-processors (about 5 minutes on a VAX 11/780 for a 256x256 image). One of our team's engineers, John Austin, has discovered a method for computing AHE on Pixel-planes that will require only a few seconds of computation for a 256x256 image. It may at first be surprising that a machine designed for efficient image generation can also be used effectively for image processing; we believe, however, that this is another indication of the usefulness and generality of Pixel-planes linear expression evaluation.

Adaptive Refinement. We have been working for some time on techniques for improving the

performance of image rendering by using machine cycles that would otherwise go wasted while a user is examining a static image on a screen. The goal is to convey as much information to the user as early as possible, with image quality constantly improving with time. A crude image is first generated rapidly; then adaptive refinements proceed where necessary as long as the user does not change viewing parameters. While this work was motivated by a desire to run such algorithms on Pixel-planes, the results are by no means limited to this machine. We have written a Technical Report (listed below), submitted for publication, that describes this work.

4. Future Work

We are currently negotiating funding for continuing work on Pixel-planes and its derivatives. We plan four major areas of development:

Architectural Innovations. We will explore two new architectural enhancements: *Pixel-powers* evaluates quadratic expressions on screen space in parallel for all pixels, just as Pixel-planes now computes linear expressions. With this capability, objects described by curved primitives, as in constructive solid geometry (CSG), can be rendered extremely rapidly. *Buffered Pixel-planes* increases the parallelism that can be achieved in our systems, and may improve the performance of our current design by a factor of 5 or more.

1.2μ CMOS Memories. Access to state-of-the-art fabrication is now becoming available at the MOSIS fabrication service, operated by DARPA/IPTO. We will therefore implement chip designs for the two architectural enhancements in this technology (1.2μ, double-metal CMOS) and couple it with our novel direct-inking packaging technology to build systems on a few boards that fit within a personal workstation.

Algorithm Development. Using our first full-scale Pixel-planes system as a test bed, we are planning a major effort in algorithm development for image generation and image processing: fast anti-aliasing, textures, direct rendering of curved surfaces, solid modeling, and medical image processing. With the arrival of Pixel-powers, we plan a joint effort with the Ballistic Research Laboratory, Aberdeen Proving Ground, MD, to develop applications for rapid display, analysis, and modification of complex solid objects.

System Building. We propose building a full-scale display that incorporates both the new architectural enhancements and advanced chip and system fabrication. Because we believe strongly that our design can be fully tested only by applying it to real applications, we plan to build two copies of an advanced graphics machine: The first will be installed in our department's graphics laboratory both for daily production use and as a platform for our algorithm development effort. The second will be installed at the Ballistic Research Laboratory to enable collaboration in developing solid modeling applications.

5. Publications

The following is a list of publications produced during the course of this project; they appear in cronological order.

Poulton, J., Fuchs, H. et al., "Pixel-Planes: Building a VLSI-Based Graphics System." *Proc. 1985 Chapel Hill Conference on VLSI*, 35-60, 1985. Included as Appendix 1 of this report.

Fuchs, H., J. Goldfeather, J. Hultquist, S. Spach, J. Austin, F. Brooks, J. Eyles, and J. Poulton., "Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-Planes". *Proc. of SIGGRAPH 85*, 111-120, 1985. Included as Appendix 2.

Goldfeather, J., and H. Fuchs, "Quadratic Surface Rendering on a Logic-Enhanced Frame-Buffer Memory System". *IEEE Computer Graphics and Applications*, **6**(1), 48-59, January, 1985.

Goldfeather, J., J. P. M. Hultquist, and H. Fuchs, "Fast Constructive-Solid Geometry Display in the Pixel-powers Graphics System". Tech. Report 86-003, Department of Computer Science, University of North Carolina at Chapel Hill.

Bergman, L., H. Fuchs, E. Grant, and S. Spach, "Image Rendering by Adaptive Refinement". Tech. Report 86-008, Department of Computer Science, University of North Carolina at Chapel Hill.

6. Personnel

Principal Investigator:
Henry Fuchs
Co-Principal Investigator:
John Poulton
Collaboration on algorithm development:
Prof. Jack Goldfeather (Dept. of Mathematics, Carleton Collge, Northfield, MN)
Engineering Staff:
John Austin (UNC-CH Microelectronic Systems Laboratory)
Wayne Dettloff (Microelectronics Center of North Carolina)
John Eyles (UNC-CH Microelectronic Systems Laboratory)
Trey Greer (UNC-CH Microelectronic Systems Laboratory)
System Software Support:
Mark Monger (UNC-CH Microelectronic Systems Laboratory)
Technical Support:
John Thomas (UNC-CH Microelectronic Systems Laboratory)
Graduate Research Assistants:
John Cromer (MS, Computer Science, May, 1986)
Justin Heinecke (MS, Computer Science, May, 1985)
Annamarie Helton
Scott Hennis (MS, Computer Science, December, 1983)
Cheng-Hong Hsieh (MS, Computer Science, May, 1985)
Jeff P. Hultquist
Susan Spach
Undergraduate Research Assistant:
Sonya Holder (BS, Physics, May, 1986)

7. Bibliography

Abram, G. and H. Fuchs., "VLSI Architectures for Computer Graphics". *Proc. NATO Advanced Study Institute on Microarchitecture of VLSI Computers*, 1984.

Bergman, L., H. Fuchs, E. Grant, and S. Spach, "Image Rendering by Adaptive Refinement". Tech. Report 86-008, Department of Computer Science, University of North Carolina at Chapel Hill.

Bishop, G. and H. Fuchs., "The Self-Tracker: A Smart Optical Sensor on Silicon". *Proc. Conf. on Advanced Research in VLSI* (MIT), 65-73, 1984.

Catmull, E., "A Hidden-Surface Algorithm with Anti-Aliasing". *Proc. SIGGRAPH 78*, 6-11, 1978.

Clark, J. and M. Hannah., "Distributed Processing in a High-Performance Smart Image Memory". *Lambda*, 4th Qtr. 1980, 40-45.

Clark, J., "The Geometry Engine: A VLSI Geometry System for Graphics". *Proc. of SIGGRAPH 82*, 127-133, 1982.

Cohen, D. and S. Demetrescu. Presentation at ACM Comp. Graph. panel on Trends in High-Performance Graphics Systems, 1980.

Deitz, P., "Solid Geometric Modeling--The Key to Improved Materiel Acquisition from Concept to Deployment". Presented at Army Operations Research Symposium XXII, Ft. Lee, VA, 3-4 Oct, 1983, and at Defense Computers-Graphics '83, Intl Conf and Expo, Washington, DC 10-14 Oct, 1983.

Deitz, P., "Predictive Signature Modeling Via Solid Geometry at the BRL". Presented at the 6th annual KRC Symposium on Ground Vehicle Signatures, Houghton, MI 21-22 August, 1984.

Deitz, P., "Modern Computer-Aided Tools for High-Resolution Weapons System Engineering". Presented at the 16th annual DOD Manufacturing Advisory Group (MTAG-84) Conference, Seattle, WA, 25-29 November, 1984.

Frisch, B. and B. Isenstein., "New Array Processor Design Solves Key Problems". To appear in *Computer Technology Review*, Spring 1985.

Fuchs, H. and B.W. Johnson, "An Expandable Multiprocessor Architecture for Video Graphics." *Proc. Sixth Annual Symposium on Comp. Arch.*, 58-67, 1979.

Fuchs, H., J. Goldfeather, J. Hultquist, S. Spach, J. Austin, F. Brooks, J. Eyles, and J. Poulton., "Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-Planes". *Proc. of SIGGRAPH 85*, 111-120, 1985.

Gharachorloo, N. and C. Pottle., "Super Buffer: A Systolic VLSI Graphics Engine for Real-Time Raster Image Generation". *Proc. 1985 Chapel Hill Conference on VLSI*, 285-305, 1985.

Goldfeather, J., and H. Fuchs, "Quadratic Surface Rendering on a Logic-Enhanced Frame-Buffer Memory System". *IEEE Computer Graphics and Applications*, 6(1), 48-59, January, 1985.

Goldfeather, J., J. P. M. Hultquist, and H. Fuchs, "Fast Constructive-Solid Geometry Display in the Pixel-powers Graphics System". Tech. Report 86-003, Department of Computer Science, University of North Carolina at Chapel Hill.

Gupta, S., R. Sproull, and I. Sutherland, "A VLSI Architecture for Updating Raster Scan Displays." *Computer Graphics*, Vol. 15, No. 3, 71-78, 1981.

Ikedo, T., "High-Speed Techniques for a 3-D Color Graphics Terminal". *IEEE Comp. Graphics and Appl.*, Vol. 4, No. 5, 46-58, 1984.

Kedem, G. and J. Ellis., "Computer Structures for Curve-Solid Classification in Geometric Modeling". *Microelectronics Center of North Carolina Tech. Rep. TR84-37*, 1984.

Parke, F. I., "Simulation and Expected Performance Analysis of Multiple Processor Z-Buffer Systems." *Computer Graphics*, Vol. 14, No. 3, 48-56, 1980.

Pinkham, R., M. Novak, and K. Guttag., "Video RAM Excels at Fast Graphics". *Electronic Design*, 161-172, 1983.

Poulton, J., Fuchs, H. et al., "Pixel-Planes: Building a VLSI-Based Graphics System." *Proc. 1985 Chapel Hill Conference on VLSI*, 35-60, 1985.

Schachter, B., "Computer Image Generation for Flight Simulation". *IEEE Comp. Graphics and Applications*, Vol. 1, No. 4, 1981.

Schumacker, R., "A New Visual System Architecture". *Proc. 2nd Annual IITEC*, 1980.

Seitz, C., A. Frey, S. Mattisson, S. Rabin, D. Speck, J. van de Snepscheut., "Hot-Clock NMOS". *Proc. of 1985 Chapel Hill Conference on VLSI*, 1-17, 1985.

Shiffman, R. and R. Parker.., "An Electrophoretic Image Display with Internal NMOS Address Logic and Display Drivers.". *Proc. of Society for Information Display*, Vol. 25, No. 2, 105-116, 1984.

Sun, E., "Graphic Processor Speeds Generation of High-Resolution Images". *Elec. Imaging*, October, 1983, 34-38.

Sutherland, I., "A Head-Mounted Three-Dimensional Display". *AFIPS*, Vol 33, Part 1, Fall Joint Computer Conference, 1968.

Sutherland, I. and R. Sproull., "A Clipping Divider". *Proc. of AFIPS*, Vol. 33, Part 1, 765-776, 1968.

Vuilleumier, R., A. Perret, F. Porret, and P. Weiss., "Novel Electromechanical Microshutter Display Device". *SID Euro-Display*, 1984.

Watkins, G., "A Real-Time Visible Surface Algorithm". Computer Science Dept., Univ. of Utah, UTECH-CSc-70-101, 1970.

Weinberg, R., "Parallel Processing Image Synthesis and Anti-Aliasing." *Proc. of SIGGRAPH '81*, 1981.

APPENDIX I

PIXEL-PLANES: Building a VLSI-Based Graphic System

John Poulton, Henry Fuchs, John D. Austin, John G. Eyles, Justin Heinecke, Cheng-Hong Hsieh, Jack Goldfeather, Jeff P. Hultquist, Susan Spach

Department of Computer Science
University of North Carolina at Chapel Hill

1. Introduction

Pixel-planes is a VLSI-based raster graphics machine that will support real-time interaction with three-dimensional shadowed, shaded, and colored images. The system's cost and complexity will be comparable to present-day line drawing systems, making it suitable for use with high-performance workstations. Potential applications include computer-aided design, medical display and imaging, molecular modeling, and simulators for flight and navigational training.

The fundamental ideas in this design have been previously published [Fuchs and Poulton, 1981; Fuchs *et al.*, 1982]. This paper reports recent progress toward building a full-scale working *Pixel-planes* system, development of a number of new graphics algorithms for the machine, and refinements in system architecture and design methods.

Much of current research in experimental graphics systems is aimed at improving the speed of image generation by dividing the

* This research supported in part by the Defense Advance Research Project Agency Contract number DAAG29-83-K-0148 (monitored by U.S. Army Research Office, Research Triangle Park, NC) and the National Science Foundation Grant number ECS-8300970.

** Department of Mathematics, Carleton College, Northfield, MN, on sabbatical at Department of Mathematics at the University of North Carolina.

display into small regions, each of which is handled by separate concurrent processors [Clark and Hannah, 1980; Gupta *et al.*, 1981; Demetrescu, 1985]. In *Pixel-planes*, this division is imbedded in a binary tree that performs the bulk of the system's computations and distributes the results to all pixels. Each pixel consists of an array of memory elements and a small processor that only performs operations local to the pixel. The heart of the system is a Smart Frame Buffer consisting of an array of identical custom chips that contain the binary tree, pixel memories and processors, and video scan-refresh circuitry. These enhanced memory chips employ a moderately dense, conventional dynamic RAM that takes up about 2/3 of the chip's silicon area; the processing circuitry takes up the remaining 1/3.

The fundamental operation of the *Pixel-planes* system is calculating linear expressions of the form $Ax + By + C$ where x and y are the coordinates of a pixel and A, B, and C are data inputs to the system. These expressions are calculated bit-serially in a binary tree multiplier/accumulator, simultaneously for all pixels. The system's hardware is not built to execute a specific set of graphics algorithms. Instead, many different algorithms can be recast into forms that evaluate linear expressions and/or require only pixel-local operations. We are continually surprised at the variety of algorithms that we and others are able to express in this form, and it is clear that the architecture is much more powerful and more general than we had first imagined.

2. Pixel-Planes Graphics System

2.1 System Overview

Figure 1 shows the relationship between the *Pixel-planes* graphics system hardware and a conventional color graphics system.

The 'front end' of the conventional graphics system is a pipeline of special processors that manipulates an image database. The database contains (typically) a list of polygons that tile the surfaces of the objects in a scene. Each polygon is described as a list of vertex coordinates (x, y, z in 'world' coordinates) and colors (values of *Red*, *Green*, *Blue* that specify the intrinsic color of the vertex). A transformation engine operates on the coordinates of the vertex list for each polygon, transforming the polygon to 'eye' coordinates in response to user input from joystick, trackball, or some similar device. Next, polygons (or portions of polygons) that are outside the viewing pyramid are clipped and perspective division is performed to transform 'eye' coordinates to 'screen' coordinates. Finally, a light-

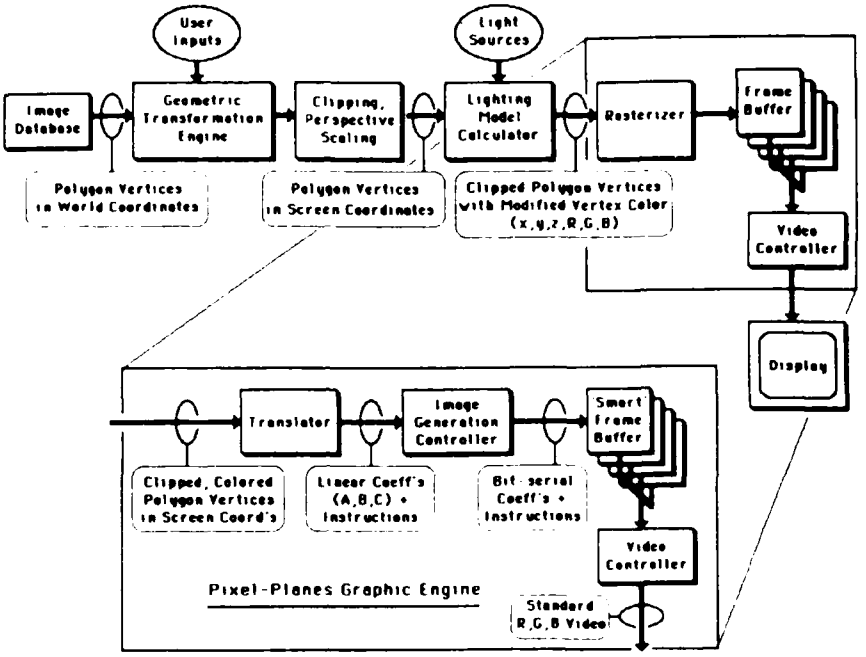


Figure 1: Pixel-Planes Graphics Engine replaces the rasterizer, frame buffer, and video controller in a conventional graphics system.

ing model calculator modifies each vertex's intrinsic color according to the position and intensity of light sources. The output of the front-end pipeline is still a list of polygon vertices, but with vertex coordinates and colors transformed to the proper value for display.

In advanced color graphics systems, the rasterizer performs a series of steps needed to translate a list of polygon vertices into a smooth-shaded, rendered, digital image, with hidden surfaces properly removed, and perhaps anti-aliased to reduce pixelization artifacts. In general, these calculations must be performed for every pixel for every polygon processed, implying massive amounts of computation and very large memory bandwidth.

The *Pixel-planes* Graphics Engine replaces the rasterizer, frame buffer, and video controller of a conventional system. Its main component is a Smart Frame Buffer composed of custom VLSI enhanced memory chips. It addresses the computational problem with a highly parallel processor that mimics a processor per pixel. The memory

bandwidth bottleneck is overcome by intimately connecting processing circuitry and memory.

2.2 Pixel-Planes Graphics Engine

The components of the Engine are:

The Translator, a special purpose micro-programmable floating-point computer, converts the scene description from a polygon vertex list into the form of coefficients A, B, C of the linear expression $F(x,y) = Ax + By + C$. It also produces an encoded instruction for each step in processing polygons or other primitives (*e.g.*, edge, z-compare, circle, paint-Red). Translation will involve, for example, describing an edge of a polygon in the form $F(x,y) = 0$, or specifying the polygon's planar surface in the form $z = F(x,y)$. In the system now under construction, the Translator is a 5 MFlop micro-programmable engine based on the Weitek 1032/1033 floating-point chip set.

The Image Generation Controller (IGC) converts the word-parallel, floating-point A, B, C coefficients from the Translator to bit-serial, 2's complement data, decodes each instruction into a stream of control words, and outputs this data and control along with the clock for the Smart Frame Buffer. Currently, the IGC is implemented as a custom chip that serializes the coefficient data and a micro-programmable control sequencer built using standard TTL parts.

The Smart Frame Buffer is organized as a series of 'logical boards', each with an array of enhanced memory chips, as shown in Figure 2. This organization reduces the bandwidth (pin-count, operating speed) necessary at the memory chip's video-data output port. Each logical board contains a 32-bit-wide register for video data, and successive logical boards are daisy-chained together to form a high-speed shift-train. Every L cycles (where L is the number of logical boards), shifting is disabled and the shift-train is loaded from a parallel set of registers on each board. While shifting is enabled, these parallel registers are loaded, one byte at a time, from selected memory chips.

Data, control, and clocks both for image generation and video output are broadcast to the enhanced memory chips. No data or control need be returned from the memories to the IGC or Video Controller, so the busses can easily be pipelined for high-speed operation.

In addition to these two uni-directional busses, a single serial scan-path links all memory chips in the frame buffer. During system

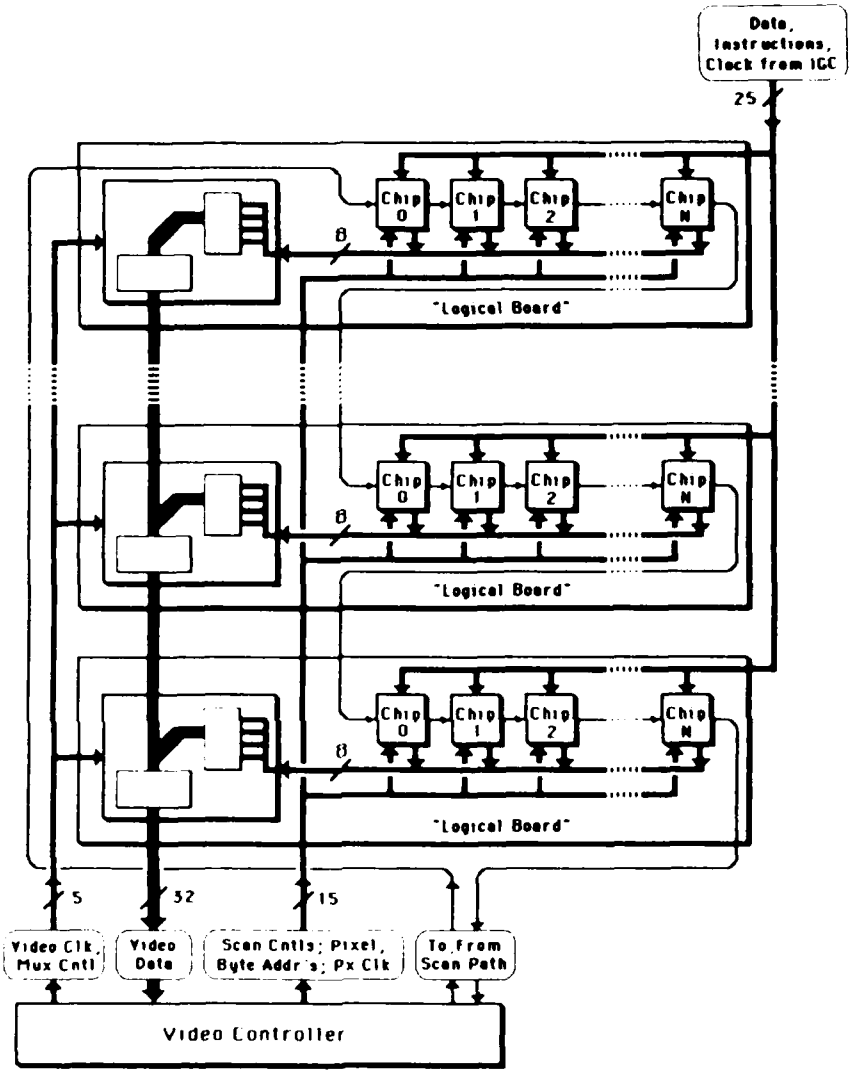


Figure 2: Pixel-Planes Smart Frame Buffer organisation.

operation, the scan-path takes the place of chip-address decoding, carrying a series of scan tokens that determine which set of memory chips is enabled for video output (only one chip on each logical board is enabled at one time). During system initialization, the scan-path is used to load various configuration registers, as discussed in Section 2.3.

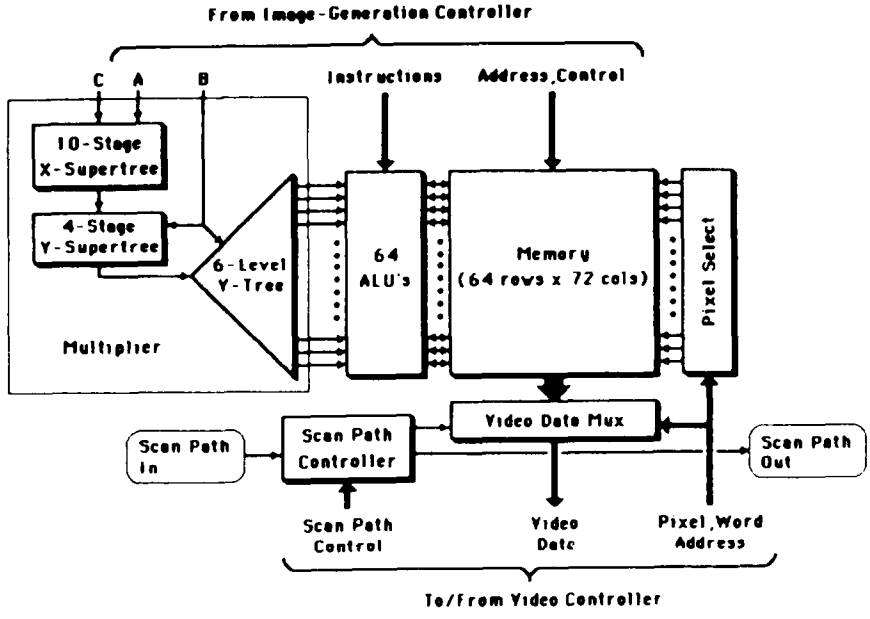


Figure 3: Block diagram of the Pixel-Planes memory chip.

The Video Controller is similar to those in conventional systems, with the exception of the token-passing method of addressing. The current version is capable of supporting a variety of display types (30 and 60 Hz, interlaced and non-interlaced, NTSC and non-standard) and any number of enhanced memory chips.

2.3 Enhanced Memory Chips

Figure 3 is a block diagram of the enhanced memory chip. It contains the Multiplier that implements the binary-tree linear-expression evaluator, an array of pixel ALU's, and a Memory system that stores data for each pixel and provides a video scan-out mechanism.

A conceptual model of a binary-tree multiplier/accumulator is shown in Figure 4. This structure is recognizable as a variation on the simple serial-parallel multiplier [Lyon, 1976], where both possible values of partial product are generated at each stage. If such a tree has N levels, and A contains K significant bits, A must be preceded by $(N-1)$ 0's; 2^N distinct values of $Ax + C$ will be generated $(0 <= x < 2^N)$, each value being $N + K + 1$ bits long.

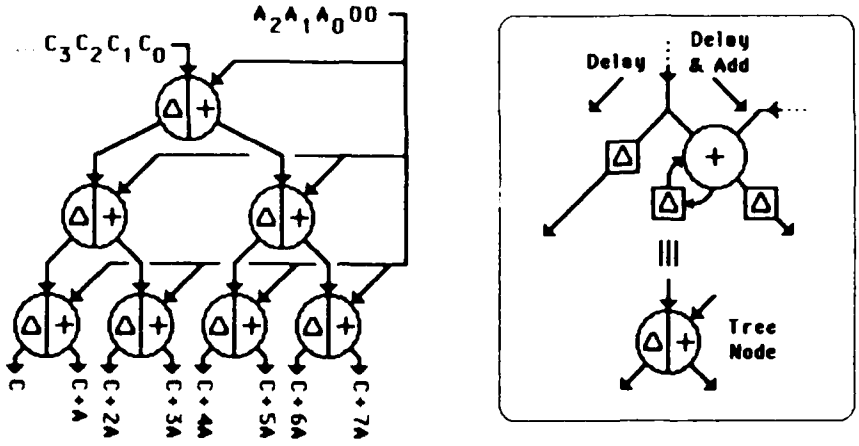


Figure 4: Conceptual model of binary-tree linear expression evaluator.

To generate the linear expression $Ax + By + C$, two binary-tree multiplier/accumulators are stacked one atop the other. For a system with 1024 x 1024 pixels, a 20-level tree is required. The top 10 levels of the tree calculate the 1024 values of $Ax + C$. The bottom 10 levels can be thought of as 1024 subtrees, each of which receives one value of $Ax + C$ as its root input, gets B as its side input, and generates 1024 values of $Ax + By + C$. For a system with N x N pixels, the binary tree requires $N^2 - 1$ multiply/accumulate stages. It performs the same function, at the same speed, as a full $2N$-stage multiplier at every pixel (requiring $2N^2 logN$ stages).

Only a small fraction of the pixels in a display can be put on a single chip, so it is necessary to break the binary tree into multiple chips. This is done by implementing a small sub-tree on each chip that covers only the pixels on the chip. A 'supertree' on each chip implements the tree levels above the sub-tree. It contains one multiply/accumulate stage for each level above the sub-tree. As shown in Figure 5, registers in the supertree are loaded at system initialization to map a path through the full tree to the local subtree. This defines the position of the chip's 64-pixel column in the full image.

It is possible, of course, to design a system without supertrees. If each chip were equipped with one extra tree node whose outputs go off-chip, the tree levels above each local subtree could be completed

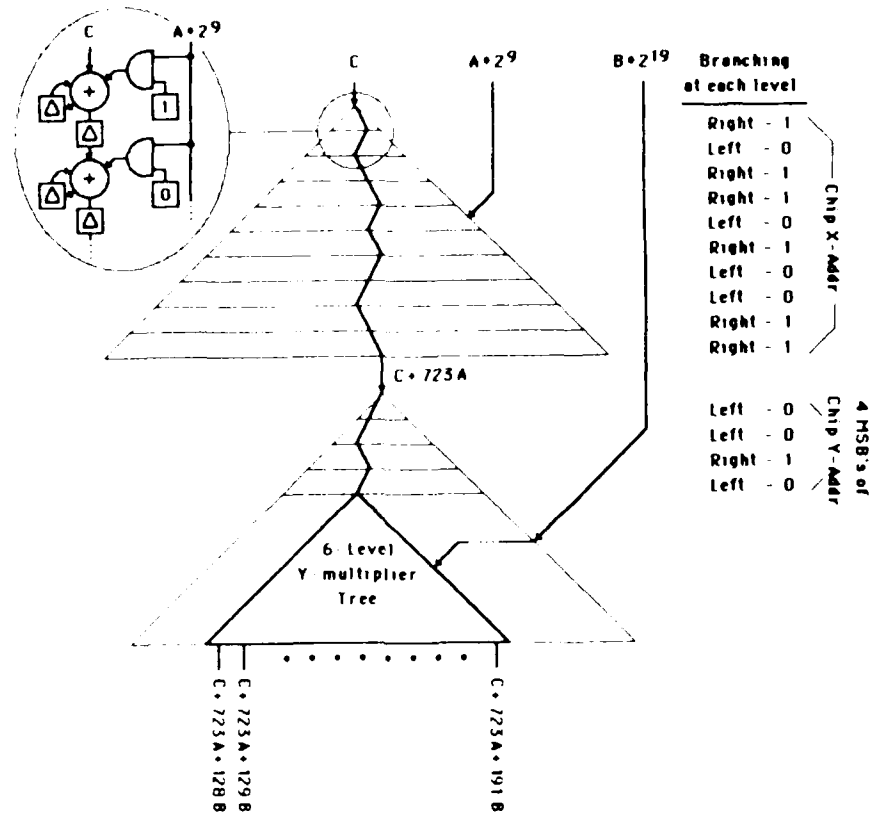


Figure 5: Supertree maps a path through full tree on each chip.

using inter-chip wiring. This external wiring would, however, reduce system speed and complicate board-level construction. The configurable supertree on our current chips has 14 levels, requiring only another 14 multiply/accumulate stages and 14 registers—a relatively modest penalty in silicon area. It also makes possible a module-redundancy scheme, described below, that supports fault tolerance in our system.

Figure 6 shows the block diagram of the ALU at each pixel. Logical operations in the ALU are performed by a one-bit adder with a multiplexer/complementer on each of its three inputs. All ALU's in the system receive function-select and register-load controls broadcast from the IGC, so the ensemble of memory chips has SIMD concurrency. The ALU contains an Enable register that con-

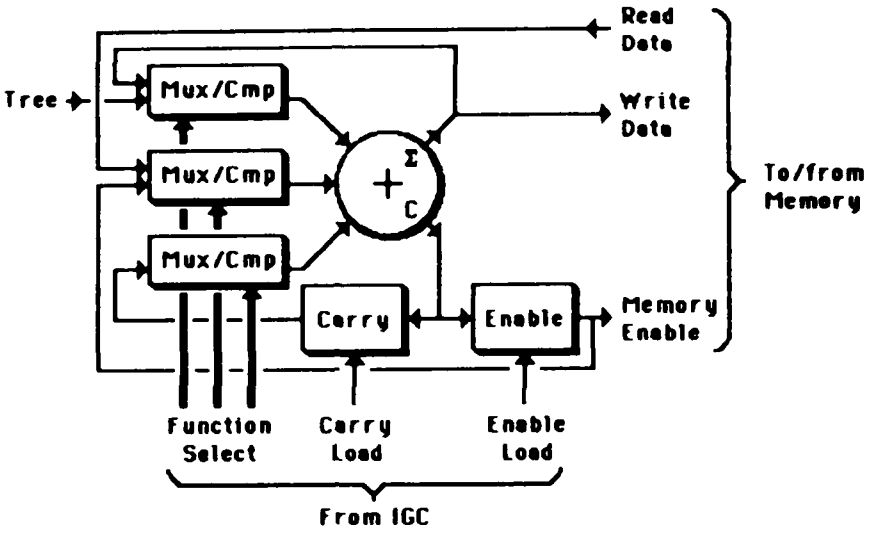


Figure 6: Block diagram of the pixel ALU.

trols memory write access, allowing each pixel to determine locally whether current memory contents can be overwritten.

The Memory system consists of a relatively dense dynamic RAM array. Each column of cells in the array contains corresponding bits in each pixel on the chip. Each row contains all of the bits in a pixel and is equipped with read/write circuitry; thus the 'word width' is extremely wide relative to a conventional memory. The memory also must provide a means for the Video Controller to access memory bits containing color-intensity information.

3. System Realisations

This section describes our experiences building several *Pixel-planes* display systems. Our first enhanced memory chip was intended as a first VLSI design exercise and not intended to become part of a working display. Two small prototype displays have been built with second- and third-generation enhanced memory chips (*Pxpl2* and *Pxpl3*), and they were sufficient to prove the basic concepts in the design. We believe, however, that it is extremely important to build a system large enough to support 'real' applications; only in this way will we convincingly demonstrate the utility of this approach to building high-performance graphics machines. We are therefore constructing a much more ambitious system using fourth-

generation chips (*Pxpl4*) that will grow to a full-scale, full-speed working display within the next year.

3.1 Pxpl2

Our second memory chip design [Fuchs *et al.*, 1982] included the local subtree for 64 pixels, a memory array with 16 bits/pixel and a single read/write port, and a simplified ALU with only a Carry generator. It lacked circuitry for the remainder of the tree, and could therefore only be used to build a 'toy' system.

A chip tester was built using a microcomputer with a parallel I/O port, the 10 chips received from fabrication were tested, and four were found to be mostly functional. The tester then became the host for a small prototype display, with Translator and Image Generation Controller functions carried out by software running on the microcomputer. The *Pxpl2* prototype verified the basic concepts in the design, executing (very slowly!) a basic set of polygon-oriented operations (polygon area definition, hidden-surface calculations using a depth-buffer, Gouraud-like smooth shading).

This exercise immediately suggested a number of design improvements:

(1) Since the memory had only a single port, image-generation had to be halted to refresh the display. This required a complex control mechanism with an external scan-line buffer to allow both image generation and video data scan-out to access pixel memory. It was clearly essential to separate these functions and to allow them to be asynchronous.

(2) Working through the details of generating separate root inputs for the sub-trees on each chip led to constructive thinking about supertrees.

(3) Several interesting algorithms had been proposed for *Pixel-planes* that would require both a more complex ALU and more memory bits/pixel.

Neither chip testing nor system operation would have been easy (or perhaps possible) if we had not first written a functional simulator for the chip. This simulator modeled all of the circuitry in the chip at the gate level, was event driven, but did not model circuit delays. It essentially captured the functional specification for the chip in an executable form. This simulator was written in a standard programming language (Pascal), a practice that we have maintained through the current version of the design (current simulators are written in C).

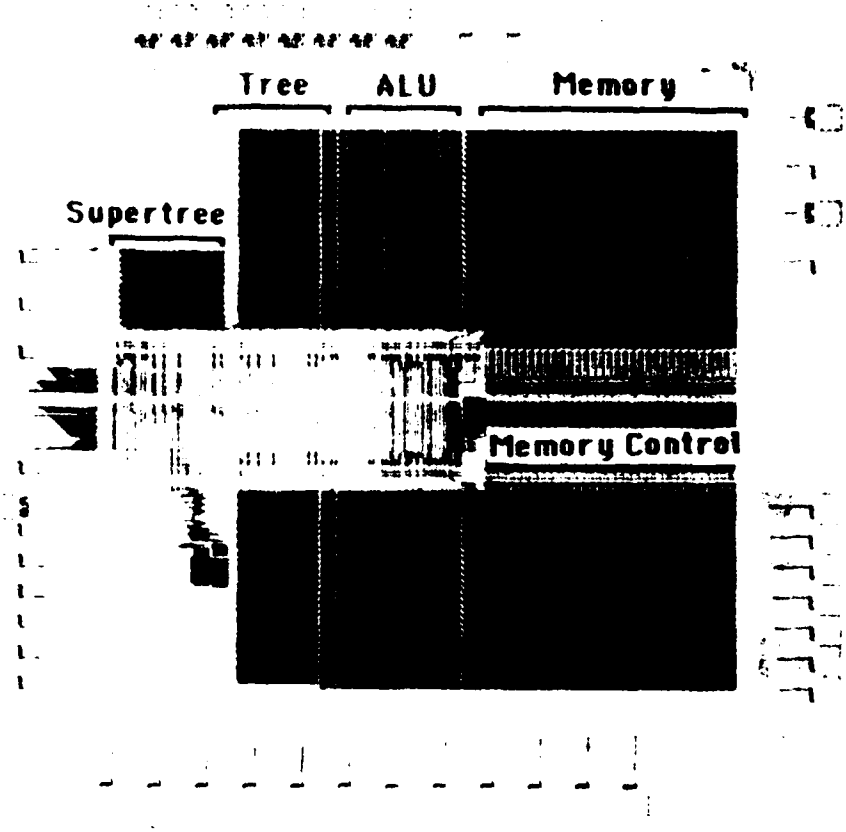


Figure 7: Photo of Pxpl3 memory chip showing major function blocks.

3.2 Pxpl3

Based on our experience with the first prototype, the next chip (Figure 7) contained many architectural improvements:

(1) A complete tree was included on each chip, implemented with the supertree notion described above.

(2) The ALU was modified to the form shown in Figure 6 to support a variety of new algorithms.

(3) Memory size was increased to 32 bits/pixel. We used a dual-ported memory cell to allow separate, asynchronous access to the pixel memory for scan refresh.

(4) Since memory access on the video-data port always proceeds in scan-line order, we installed a pixel-addressing mechanism that uses serial-shift tokens. A 'global' token that passes from chip to chip performs chip select, while a 'local' token register inside each chip manipulates the pointer to the currently selected pixel. The token-addressing scheme reduces chip pin-count significantly, and is a faster mechanism than conventional address decoders.

(5) Since a serial shift-path was already needed to support the global token mechanism, we elected to make multiple use of this path. During system initialization, this inter-chip path can be diverted on each chip into the 'configuration' register that programs the supertree, thus linking all configuration registers into one large scan-path.

(6) Reasonable yield from fabrication at 4 micron feature size allows only 64 pixels on a chip (1.5 micron feature size would allow a few hundred pixels per chip). Current fabrication limits led us to investigate other ways of getting more memory on a single packaged device. We saw that in future chip implementations, the 64-pixel chip might become merely one of a number of modules on a much larger chip, where some modules are allowed to be faulty. An Alive register was installed on the chip to provide a way of turning off faulty modules under software control. On initialization, these registers can, like the configuration registers, be linked together by the serial scan-path. A pattern of 1's and 0's scanned in corresponding to good and faulty modules. Modules (chips) with Alive set to 0 are disabled for video output, and their configuration registers are disconnected from the scan-path during supertree programming.

As in the *Pxpl2* prototype, a complete functional simulator was written for each of the image-generation functional blocks, the Translator, IGC, and Frame Buffer. This simulator could produce crude images to help check the correct operation of various algorithms. The simulators for the Translator and IGC, with slight modifications, became the driver programs for the actual hardware.

Chip testing was done essentially on the display system itself. Since the memory chips are intended to produce graphics images, we simply plugged a single chip into the prototype display, exercised its functions, and observed the results on a color monitor where groups of memory bits were interpreted as color intensities. This rather crude testing strategy was surprisingly effective, even in diagnosing design faults.

Testing revealed several problems with the design:

(1) Over-aggressive use of the newly-available buried contacts in the memory (design rules for burieds were still rather vague) was most likely responsible for rather poor yield (approximately 20%).

(2) The dual-ported memory cell design was flawed and failed to decouple the two ports fully. Image-generation and video scan clocks therefore had to be synchronized.

(3) Failure to carry through a rigorous timing analysis of the memory system and its video output circuitry led to a timing fault that drastically reduced scan-out speed (approximately 1 MHz), but still allowed the chip to function. Under this limitation, the prototype display could be populated only with eight chips per logical board.

The system works correctly under restrictions imposed by the design flaws. Its speed is limited not by hardware design problems, but by the software that emulates the Translator and IGC. Since this software runs about 1000 times slower than the on-chip processors, the system is fast enough to produce only very crude animation (about 2-3 updates per second on an image with 6 polygons).

The module-level fault tolerance scheme using the Alive register was successfully tested on the *Pxpl3* prototype. In fact, the entire serial-shift mechanism for Alive, supertree configuration, and global-token passing worked successfully on first silicon.

Building and testing the *Pxpl3* prototype brought forcefully to our attention the need to build hardware to execute the Translator and IGC functions. The experience also suggested three important design changes in the memory chip:

(1) The fabrication yield for the *Pxpl3* chips would have been greatly improved (better than 2x) with the addition of a redundant memory column and a redundant row.

(2) The dual-ported memory scheme did not appear to be a very effective way to support scan refresh, even had it been successfully implemented. It provides much higher bandwidth in the second port than is required by the scan-out process and requires a memory cell about twice the size of a conventional cell.

(3) Since the multiplier tree in *Pxpl3* is implemented essentially as shown in Figure 4, the tree must be flushed after the formation of each result, in order to clear the carry registers at each node.

A 30% speedup could be achieved if the multiplier were more fully pipelined.

3.3 *Pxpl4*

The improvements suggested by the *Pxpl3* prototype have been built into a new enhanced memory chip (*Pxpl4*), in fabrication at the time of writing. The chip contains 64 pixels, each with 72 bits of memory. In 4-micron nMOS, active circuitry (excluding pad frame and wiring) is 7.5 x 4.0 mm and contains about 33,000 transistors. Of this area, about 70% is devoted to memory, 20% to the binary-tree circuitry, and 10% to the pixel ALU. With MOSIS's 3-micron fabrication, two modules (128 pixels) can be built inside a MOSIS-standard pad frame.

The system built around this chip will be expandable to 512 x 512 pixels with 72 bits/pixel (or it can display 1024 x 1024 pixels with 18 bits/pixel). This system will be hosted by a high-performance workstation that will store and manipulate image data-bases, provide user interaction, and initially carry out part of the polygon transformation tasks in scene generation. (Later versions of the system will perform transformations using special hardware, such as the Geometry Engine in the Silicon Graphics IRIS [Clark, 1982]).

The following paragraphs detail various design enhancements in the current memory chip.

Multiplier Pipelining

Multiplier operations are fully overlapped by including a small amount of additional hardware for a pipeline register. Figure 8 shows the details of this scheme, which differs somewhat from that in conventional pipelined multipliers [Lyon, 1976]. The pipelining register is 'sticky': when it receives a logic-1 it is locked into this state until a global clear is generated. Thus, a stream of 1's marches down the multiplier just behind the formation of partial products contributing to the MSB of the result, and just ahead of the LSB of the new constant coefficient. When the stream of 1's reaches the last stage, a clear is generated that simultaneously re-enables multiplication at all stages.

Memory Design

Pxpl4 uses a 4-transistor dynamic memory cell that has the useful property of refreshing itself during a read operation. Since each memory row is connected only to its pixel ALU, no special sense amplifier is needed for read access; simulations show that the memory operates faster (about 20 MHz) without one.

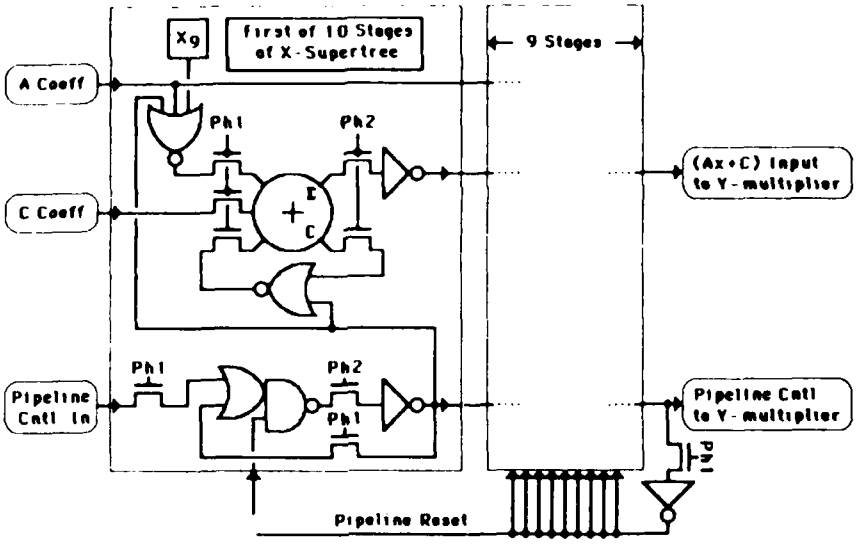


Figure 8: Circuitry for pipelining tree operations. The X-supertree is shown, but the scheme is used in the Y-tree as well.

The video output port of the pixel memory is implemented as a single double-buffered register per chip, the Shadow Register, in which a copy of the currently selected pixel's memory is built up sequentially. The scheme is shown in Figure 9.

A pixel selector points at the pixel (memory row) needed for the next scan-line and puts a copy of the data from each bit onto a one-bit bus during each read or write cycle. Simultaneously, the memory address decoder output is delayed and used to load data from this bus into the element of the shadow register corresponding to the selected memory bit. Thus, as each bit of memory is 'visited' during image generation, it is copied into the master half of the Shadow Register. At the end of a scan-line, the Video Controller unloads the master into the slave of the Shadow Register, where the data is available for output. The Shadow Register mechanism is much more space-efficient than a full dual-ported memory. It requires some care, however, in design and in operation to avoid data corruption due to synchronization failure and to ensure that image-intensity bits are visited often enough to update the register once per scan-line. Neither of these problems is difficult to overcome in practice.

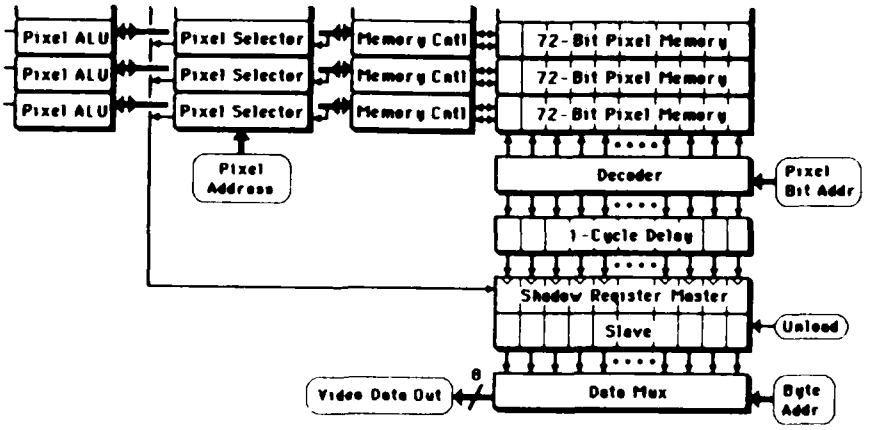


Figure 9: Video memory port implemented as a Shadow Register.

Redundant Modules and Circuits

Pxpl4 retains the Alive mechanism for module fault tolerance that was tested in *Pxpl3* and adds circuitry to support redundant memory elements to make each module more robust.

The chip contains one extra memory column. A redundant-column address register is added to the chip's configuration register, so that the address of the column to be replaced can be scanned in during system initialization.

Provision of a redundant row is somewhat more difficult, since one of the ALU-memory interfaces must be re-connected to the redundant memory row. Re-connection cannot be readily implemented without undue loss in system speed, so instead we provide an entire extra pixel with ALU and complete sub-tree path. The 6 nodes of local tree above the redundant pixel are realized simply by building a full 20 stages of supertree. The configuration (address) registers in these stages contain the address of the redundant pixel, and are loaded with the rest of the address at initialization. Redundant row and column enables are also provided to turn the entire mechanism off.

The redundant column circuitry requires only about 1.4% of the total active circuit area and the redundant row about 5.3%.

3.4 Buffered Pixel-Planes

One drawback of our present system is that the full parallelism

cannot be utilized subsequent to scan-conversion. During visibility and painting calculations, all pixels outside the currently processed primitive are idle.

We are investigating an alternative system design, called 'Buffered Pixel-Planes', that improves parallelism. A modified Image Generation Controller with accept/reject circuitry and a FIFO is fully integrated onto a custom chip, and many copies are distributed across the system, each supervising a group of enhanced memory chips. The Translator sends bounding-box data for each primitive ahead of its coefficients and instructions. Each IGC accepts or rejects the current primitive based on the bounding box; if inside, coefficients and instructions are accepted and pushed into the FIFO for processing.

We have simulated the behavior of such a system processing images of moderate complexity (up to 1000 polygons), and we predict approximately 5-fold speedup with modest (10-polygon) FIFO size.

4. Design Methodology

4.1 Tools

For the nMOS realization of our current chips, we use mask-level layout, layout analysis, and circuit simulation design tools distributed by the University of California at Berkeley.

We have written in C the logic-gate-level simulators for the memory chip and for other system components. These simulators are used first to check the correctness of the logic design for the system, then to generate test vectors for switch-level simulation of the chip circuitry.

Most of the design of the custom chips was done by two designers working on a Digital Equipment VAX 11/750 minicomputer with two Lexidata 3700 color displays.

The lack of well-integrated design tools that go smoothly from silicon design to board design is a serious impediment to our work. Board-level logic design and analysis are still done using paper and pencil, with considerable assistance from standard UNIX program-development tools. Boards are layed out with a graphic chip-layout editor and fabricated using MOSIS's PC-board service.

For some time we have been working on a CMOS version of the enhanced memory chip. Mask-level design of CMOS projects is unattractive for two reasons: First, the additional complexity of CMOS technology makes an already-difficult layout task much more

tedious. Second, the fabrication technology is developing rapidly, and it is not clear that scalable design rules for mask layout will be an effective way of tracking these advances. We have therefore been using (and assisting in the development of) the VIVID* symbolic-layout design system [Rosenberg *et al.*, 1985]. The system includes a hierarchical layout compactor that translates symbolic layout to mask with the help of a technology file that captures all relevant information about a particular fabrication process. In this way a given symbolic design can hope to survive considerable change in the target fabrication technology.

4.2 Design Style

Constructing a full-scale, full-speed system is a much more complex task than building a small prototype. The principal lesson learned from our early prototype construction was the need for complete documentation and precise interface specifications. We have therefore adopted for all system components a design style whose elements are:

(1) The system is decomposed into modules following a restricted hierarchy, in which only leaf-cells are allowed to contain circuit elements. The hierarchy is maintained in parallel in the physical domain (*e.g.*, chip layout) the logical domain (*e.g.*, logic schematics), and the behavioral domain (*e.g.*, simulators that model the logic). Composition cells may contain only interconnection information (abuttment, for example, within a chip layout) and other cells. Leaf cells may contain only circuit elements (logic gates in the logical description; transistors, wires, contact cuts in the physical description).

(2) Borrowing from strongly-typed programming languages, we impose a strong-typing scheme on all signals in the system. To ensure that modules are 'correctly connected' (*e.g.*, timing conventions and active-levels are observed), only a few signal types are allowed for connection between modules. The typing scheme is based on non-overlapping multi-phase clocks, and if applied carefully, avoids race conditions in sequential circuits. The signal types are encoded in a suffix attached to every signal name, providing a powerful documentation aid.

(3) Special hazards are involved where clocking convention (*e.g.* edge-triggered vs. level-sensitive latching) and implementation

* VIVID is a trademark of the Microelectronics Center of North Carolina

technology (TTL logic vs. custom MOS) changes, particularly at the chip I/O pads. To help assure that this interface will work properly, we define its timing conventions in a simple way, using two-sided timing constraints.

(4) Every major module in the design is modeled by a functional simulator. The simulated modules are tested separately, then plugged together to check the correctness of interfaces and overall operation of the simulated system. The simulators provide test vectors for chip simulation and testing.

The signal-typing/timing schemes are similar to [Noice *et al.*, 1982] and [Karplus, 1984]. Other elements of the style were influenced by [Lattin *et al.*, 1981; Stefik and Conway, 1982; Stefik *et al.*, 1982], among other sources.

4.3 Clocking Techniques

Our nMOS custom chips use a high-voltage clocking scheme ('hot clocks') suggested to us by [Seitz, 1982] and described in [Seitz *et al.*, 1985]. The main advantage of the technique is that n-enhancement transistors transmit a logic-HI without threshold drop and at much higher speed. In general, this clocking method produces layouts that are denser and much faster than conventional single-supply designs.

In a system with many custom chips, it is extremely inconvenient to generate these clock signals off-chip at a non-standard voltage. We have therefore built on-chip clock drivers that perform level translation and single-to-2-phase conversion. A separate input pin, biased typically at 8 volts, powers only these circuits. We have successfully built and tested a number of such high-voltage drivers, and our current design charges 100 pf to 7 volts in about 10 nsec.

The clock signals are produced in a single generator on the chip and distributed, so far as possible, continuously in metal wiring. For routing purposes, the clocks are second in importance only to Vdd and ground. For the inevitable cross-unders, we use 'low resistance wire', essentially an extended buried contact whose sheet resistance we have measured at about 7-8 ohms/square.

Level-sensitive register controls require qualified clocks that are generated in clocked, bootstrapped drivers [Joynson *et al.*, 1972]. The design of these compact drivers is not difficult, and they can be made to generate qualified clocks that follow the primary clock signal with nearly zero delay.

Polygon input data:
A_i, B_i, C_i for each edge i

For each edge i define:
$F(x,y) = A_i x + B_i y + C_i$

Pixel at (x,y) is inside polygon
if and only if:
$F_i(x,y) > 0$ for all i

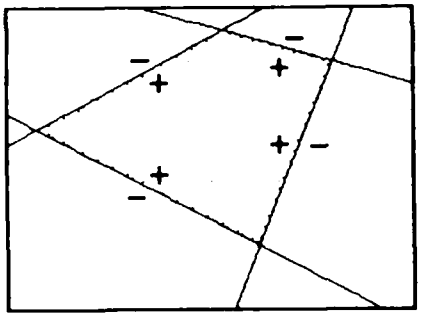


Figure 10: Scan-coverting polygons using linear expressions.

5. *Pixel-Planes Algorithms*

In this section we briefly describes how polygonal images are processed in *Pixel-planes*, and we outline several new algorithms, more fully described in [Fuchs *et al.*, 1985]. The timing estimates in this section assume that the *Pxpl4* chips are clocked at 10 MHz.

Rendering smooth-shaded polygons requires scan conversion, hidden surface elimination, and shading calculations:

Scan conversion is outlined in Figure 10. Processing begins by enabling all pixels in the display. Edges are encoded in linear equations of the form $F(x,y) = Ax + By + C = 0$, and the sign of F is tested at every pixel to determine visibility. Scan conversion leaves pixels outside the current polygon disabled; only those inside participate in further visibility and shading calculations.

Hidden surface elimination can be performed using a depth-buffer algorithm in which the z-coordinate of a pixel is encoded in a set of coefficients A, B, C by the linear expression $z = Ax + By + C$. Each pixel stores a value of z for the closest polygon so far processed and compares this value with the incoming z. If the new z is closer, the current polygon is visible at this pixel, and it remains enabled for shading, updating its z-buffer. If the stored z is closer than the new z, the pixel is disabled during shading.

Smooth shading is accomplished by computing a set of coefficients for each of R, G, and B, so that the color-intensity at each pixel is approximated by $F(x,y)$. Gouraud-like smooth shading can be carried out by painting each multi-sided polygon as a series of triangles (scan-conversion and hidden surface elimination,

however, need only be done once for each polygon).

Polygon processing time depends on the number of edges and the number of bits needed to represent the function $F(x, y)$ for each operation. Approximately 30,000 4-sided polygons of arbitrary shape and orientation can be processed per second, using the steps outlined above.

Shadows are important depth cues in interactive systems, and we have developed a method, similar to [Brotman and Badler, 1984], for casting shadows from arbitrary light sources using using shadow volumes [Crow, 1977]. For each polygon in the image, the set of visible pixels that lie in the frustum of the polygon's cast shadow are determined, and the color intensity of these pixels is diminished by an appropriate factor. Shadows are post-processed after a non-shadowed polygon image has been generated. The shadows for approximately 78,000 polygons can be computed per second.

Filled circles can be rendered rapidly in *Pixel-planes* by treating a circle as a polygon with one edge. The method separates the equation of a circle into a linear part that differs for each circle size and position, and a quadratic part that is the same for all circles. The quadratic part is pre-computed and its distinct values are loaded into every pixel at system initialization. Circles are processed by encoding center-position and radius in coefficients A, B, C and adding the linear expression to the stored quadratic term at each pixel. This method can readily be extended to render the other conic sections, such as ellipses. Spheres can be approximated by a quadratic surface, depth-sorted using a Z-buffer, and highlighted from an arbitrary light source. Approximately 34,000 spheres can be processed per second.

Texture mapping can be performed by using the linear expression evaluator to compute a texture plane address at every pixel. The appropriate color value for a pixel is then looked up in a texture table, transmitted entry by entry to the Smart Frame Buffer.

Anti-aliasing may be accomplished by one of two methods. The first, similar to 'super-sampling', blends a newly computed image with a previously computed image in a series of steps that successively refine the image. To support rapid interaction, the image is only refined when stationary. A second approach uses a method similar to that used on the Evans and Sutherland CT-5 real-time image generation system [Schumacker, 1980]. This method assumes that a visibility ordering of the polygons has already taken place, and uses a sub-pixel coverage mask to compute the anti-aliased image.

Transparency effects can be produced using the sub-pixel cover-

age mask for successive refinement, or by disabling patterns of pixels (*e.g.* a checkerboard) during polygon processing.

Adaptive Histogram Equalization (AHE) [Pizer *et al.*, 1984] is a powerful image processing technique used for grey level assignment and contrast enhancement of Computed Tomographic (CT) images. A local histogram is computed for every pixel in the image, and then used to compute a new grey level assignment for that pixel. For a 512 x 512 image, this method requires about 5 minutes of computation on a VAX 11/780, and is therefore too inefficient for most uses. The parallel processing power of *Pixel-planes* can be used to compute simultaneously the grey level assignment for each pixel in the image, without the need for histogram calculation. A rank counter, maintained in a portion of each pixel's memory, can be incremented using the pixel ALU. The intensity of a given pixel is broadcast and compared, in parallel, to the intensity of all pixels that are within a local region. The rank counter is incremented at all pixels in the local region whose intensity is greater than the given pixel. After all pixels have been processed, the rank counter values are scaled and displayed. We estimate a 512 x 512 image will require approximately 4 seconds to compute on *Pixel-planes*.

6. Comparison with Other Architectures

We divide alternative VLSI-based architectures for graphics into two classes (as outlined in [Abram and Fuchs, 1984]): those that divide the image plane into sub-planes, with a processor for each subdivision, and those that divide the object database, assigning a processor to each subdivision. The *Pixel-planes* system is an example of the former, and we therefore compare it with two other systems of this type.

6.1 Architectures for Image-plane Subdivision

Several groups ([Fuchs and Johnson, 1979]; [Clark and Hannah, 1980]; [Gupta *et al.*, 1981]) have proposed systems that make more effective use of commercial RAM chips than conventional frame buffers; we refer to this as the *interlaced* approach. In [Clark and Hannah, 1980], the RAM's are interlaced so that on any 8 x 8 area of the screen, one pixel comes from each of the RAM's; each memory contains every eighth pixel in every eighth row. The scheme uses two layers of special processors organized in columns and rows, with a row-processor in charge of each RAM chip (or group of RAM chips when more than 1 bit/pixel). An entire 8 x 8 patch on the screen can be accessed with a single memory reference by the 64 row processors,

so a polygon (or other primitive) roughly the size of a patch, or larger, can be processed with considerable parallelism.

A major advantage of the *interlaced* approach is that it uses high-density commercial RAMs and yet achieves performance greatly improved over conventional frame buffers with relatively few custom chips. This design is hampered, however, by the bandwidth limitations imposed by separating memories and processors onto separate chips.

Another recent approach, described in [Demetrescu, 1985], employs 'Scan-Line Addressable Memories' (SLAM's). A system with 1024 x 1024 one-bit pixels is organized in 16 rows, each with a Scan-Line Processor in charge of 4 SLAM chips. Each of these units contains and controls all of the pixels in 64 successive scan-lines. Each SLAM chip contains a conventional RAM array, organized as 64 rows of 256 one-bit pixels, augmented with an array of very simple processors that operate in parallel on all pixels in a row. In one cycle of operation, all pixels in 16 scan lines can be accessed. The Scan-Line Processors provide buffering of graphics primitives, so that very high parallelism can be achieved.

The system-level implementation of a SLAM-based display should be very clean. In contrast to the *interlaced* design, high-bandwidth memory-processor communication wiring is completely encapsulated in the SLAM chips. Commands and data are broadcast from each Scan-Line Processor to its SLAM's over a low-bandwidth bus. The SLAM design solves the display-refresh problem without interrupting image processing (by including a display shift register on the SLAM chip). These are the principal features common to the SLAM and *Pixel-planes* approaches.

6.2 Comparison with Pixel-planes

For today's high-performance workstations, where the display requires one or a few bit-planes and handles (mainly) multiple windows with text, lines, and flat-shaded polygons, the SLAM approach is extremely attractive. For such applications, it appears to be considerably faster than either the *interlaced* or *Pixel-planes* designs and is several orders of magnitude faster than conventional frame buffers. The cost of the approach, like ours, is the need to use custom-designed memory chips. The processors on the SLAM chip are extremely simple and appear to require very little area, however, perhaps as little as 1/10 that of our processors.

The *Pixel-planes* system is targeted at applications more demanding than the displays in current workstations, such as medical

display and imaging, molecular modeling, mechanical design systems, and flight and navigational simulators. These applications require interaction with 3D images needing visibility determination, smooth shading, shadows, and textures; images with perhaps thousands of primitives and significant depth complexity must be updated at frame rates.

Methods for improving perceived image quality necessarily rely on storing additional information at each pixel. Clearly, the most effective means of improving performance is accessing and processing this data in parallel, closely associating a large amount of pixel memory with a pixel processor. The *Pixel-planes* design provides the power of a processor per pixel, at relatively modest cost in silicon area, and a very general method for computing images.

The *interlaced* approach cannot grow gracefully in the dimension of bits/pixel because of chip I/O limitations. In the SLAM design, one alternative for growth in this direction is multiple banks of SLAM, one for each bit plane. To expand such a system to the size accommodated by the current *Pixel-planes* chip would entail a copy of the processor for each of 72 bit-planes, an intolerable increase in silicon area. The other alternative is using a column in the SLAM to hold all bits of a pixel, then bit-serially processing data; this alternative is similar to our approach, but it fails to provide a very general image-computation method.

For applications that require accessing large amounts of memory per pixel, our system should be denser and faster than either of the other approaches. In effect, we have already paid the price of accessing many bits/pixel: bit-serial data access and a more general (and costly) method of display refresh.

7. Acknowledgements

We wish to thank Vernon Chi (Director), Mark Monger, and John Thomas, of the UNC Microelectronic Systems Laboratory, for design and technical assistance in building the *Pixel-planes* system. We also wish to thank Alan Paeth and Alan Bell of Xerox Palo Alto Research Center for collaborating in the design of *Pxpl2* and *Pxpl3*, Scott Hennes for assistance with the *Pxpl3* chip, Fred Brooks for the basic circle scan-conversion algorithm, and Turner Whitted for discussions about anti-aliasing and transparency algorithms.

8. References

Abram, G. D. and H. Fuchs. July, 1984. "VLSI Architectures for Computer Graphics," *Proceedings of the NATO Advanced*

Study Institute on Microelectronics of VLSI Computers, Sogesta-Urbino, Italy.

Brotman, L. S. and N. I. Badler. October, 1984. "Generating Soft Shadows with a Depth Buffer Algorithm," *IEEE Computer Graphics and Applications*, 5-12.

Clark, J. H. and M. R. Hannah. 4th Quarter, 1980. "Distributed Processing in a High-Performance Smart Image Memory," *LAMBDA*, 40-45. (LAMBDA is now VLSI Design).

Clark, J. H. July, 1982. "The Geometry Engine: A VLSI Geometry System for Graphics," *Computer Graphics*, **16**(3), 127-133. (Proc. Siggraph '82).

Crow, F. C. July, 1977. "Shadow Algorithms for Computer Graphics," *Computer Graphics* , **11**(2), 242-248. (Proc. Siggraph '77).

Demetrescu, S. May, 1985. "High Speed Image Rasterization Using Scan Line Access Memories," *In these Proceedings*.

Fuchs, H. and B. Johnson. April, 1979. "An Expandable Multiprocessor Architecture for Video Graphics," *Proceedings of 6th ACM-IEEE Symposium on Computer Architecture*, 58-67.

Fuchs, H. and J. Poulton. 3rd Quarter, 1981. "Pixel-planes: A VLSI-Oriented Design for a Raster Graphics Engine," *VLSI Design*, **2**(3), 20-28.

Fuchs, H., J. Poulton, A. Paeth, and A. Bell. January, 1982. "Developing Pixel Planes, A Smart Memory-Based Raster Graphics System," *Proceedings of the 1982 MIT Conference on Advanced Research in VLSI*, Dedham, MA, Artech House, 137-146.

Fuchs, H., J. Goldfeather, J. P. Hultquist, S. Spach, J. D. Austin, J. G. Eyles, and J. Poulton. January, 1985. Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-Planes, Technical Report 85-002, Dept. of Computer Science, University of North Carolina at Chapel Hill.

Gupta, S., R. F. Sproull, and I. E. Sutherland. August, 1981. "A VLSI Architecture for Updating Raster Scan Displays," *Computer Graphics*, **15**(3), 71-78. (Proc. Siggraph '81).

Joynson, R. E., J. L. Mundy, J. F. Burgess, and C. Neugebauer. June, 1972. "Eliminating Threshold Losses in MOS Circuits by Bootstrapping Using Varactor Coupling," *IEEE Journal of Solid-State Circuits*, **SC-7**, 217–224.

Karplus, K. August, 1984. A Formal Model for MOS Clocking Disciplines, Technical Report 84-632, Dept. of Computer Science, Cornell University, Ithaca, NY.

Lattin, W. W., J. A. Bayliss, D. L. Budde, J. R. Rattner, and W. S. Richardson. 2nd Quarter, 1981. "A Methodology for VLSI Chip Design," *LAMBDA*, **2**(2), 34–44. (LAMBDA is now VLSI Design).

Lyon, R. F. April, 1976. "Two's Complement Pipeline Multipliers," *IEEE Transactions on Communications*, **COM-24**, 418–425.

Noice, D., R. Mathews, and J. Newkirk. 1982. "A Clocking Discipline for Two-Phase Digital Systems," *Proc., IEEE International Conference on Circuits and Computers*, 108–111.

Pizer, S. M., J. B. Zimmerman, and E. V. Staab. 1984. "Adaptive Grey Level Assignment in CT Scan Display," *Journal of Computer Assisted Tomography*, **8**(2), 300–305.

Rosenberg, J. B., C. D. Rogers, and S. Daniel. 1985. "An Overview of VIVID, MCNC's Vertically Integrated Symbolic Design System," *To appear in the Proceedings of the 1985 Design Automation Conference.*

Schumacker, R. A. November 1980. "A New Visual System Architecture," *Proceedings of the 2nd Annual IITEC*, Salt Lake City.

Seitz, C. 1982.Private Communication.

Seitz, C. L., A. H. Frey, S. Mattisson, S. D. Rabin, D. A. Speck, and J. L. A. Snepscheut. May, 1985. "Hot-Clock nMOS," *In these Proceedings.*

Stefik, M., D. Bobrow, A. Bell, H. Brown, L. Conway, and C. Tong. January, 1982. "The Partitioning of Concerns in Digital System Design," *Proceedings of the 1982 MIT Conference on Advanced Research in VLSI*, Dedham, MA, Artech House, 43–52.

Stefik, M. and L. Conway. April 28, 1982. Toward the Principled Engineering of Knowledge, KB-VLSI-82-18, Xerox, Palo Alto.

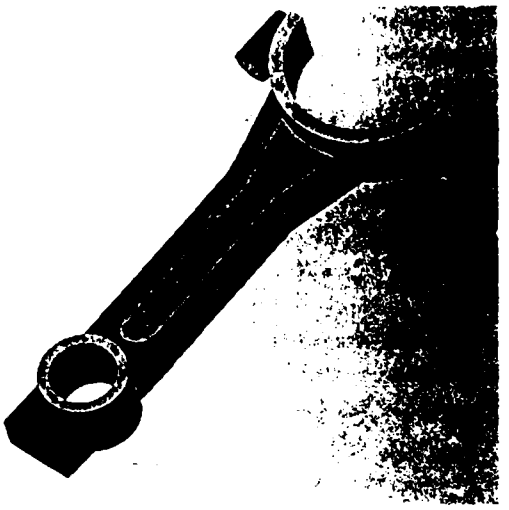
A custom VLSI-based system offers rapid rendering of elaborate, curved objects defined by constructive solid geometry, paving the way for real-time interaction.

Quadratic Surface Rendering on a Logic-Enhanced Frame-Buffer Memory

Jack Goldfeather Carleton College

Henry Fuchs

University of North Carolina at Chapel Hill



A new system, Pixel-powers, has been designed for the rapid rendering of curved surfaces. This system is a generalization of the design of our logic-enhanced frame-buffer memory system, Pixel-planes. Our new design can directly evaluate quadratic expressions of the form $Ax^2 + Bxy + Cy^2 + Dx + Ey + F$ at every pixel (x, y) in the image. Parallel-Sample images generated by a high-level emulation of the new system are shown.

The Pixel-planes system was designed to generate 3D, smooth-shaded polygonal images rapidly enough to support real-time interaction.[1-3] The system design takes advantage of the fact that many of the calculations necessary to generate a polygonal raster graphic image (polygon scan conversion, z-buffer visibility testing, and Gourand shading) often are linear in the pixel coordinates. The design incorporates a tree of adders to compute expressions of the form $Ax + By + C$ simultaneously at each pixel (x, y).

In essence this tree, which we shall call the Linear Expression Evaluator (the "multiplier tree" in previous reports), receives the three bit streams A, B, and C as input, and distributes the calculations for the linear expression $Ax + By + C$ in terms of the binary representation of the pixel coordinates (x, y). With our colleagues, we have built three generations of small prototypes of this system, and have developed image-generation algorithms for them.

This article reports a major enhancement to the Pixel-planes design, for directly handling second-order curved surfaces as well as planar ones. This enhanced system also appears to generalize to still higher order surfaces. We call the new system Pixel-powers.

Pixel-powers has a more elaborate tree structure than Pixel-planes, one that can directly evaluate expressions of the form $Ax^2 + Bxy + Cy^2 + Dx + Ey + F$ for every pixel (x, y) simultaneously, when the coefficients A, B, C, D, E, and F are input directly to the enhanced tree structure. We call this module the Quadratic Expression Evaluator. Briefly, the QEE is constructed by linking two LEE's together with some additional delays and adders.

With this capability, Pixel-powers should be able to generate elaborate, smooth-shaded, curved objects defined, for instance, by constructive solid geometry (CGS) in real time. The primitive objects (cylinders and spheres, for example) are calculated efficiently with the enhanced tree structure, and a one-bit ALU at each pixel calculates the logical combinations (union, intersection, difference) of these primitive objects.[4]

We estimate that Pixel-powers will have a 35 percent greater chip area than Pixel-planes, but should run at the same clock speed. We estimate that the presently running (10-MHz) Pixel-planes chips yield a system capable of generating approximately 30,000 smooth-shaded, full-color

0272-1716/86/0100-0048$01.00 © 1986 IEEE

polygons per second, including z-buffer visibility computations.

We do not yet have a precise estimate of the speed of a 10-MHz Pixel-powers system, but our functional simulator generates images of an internal combustion engine connecting rod in 900 μs (simulated time) assuming the rest of the system can keep up (a difficult task).

It is important to note in passing that, although CSG representations are widely used,[4] there have been only a few custom VLSI-based designs for them.[5]

The Linear Expression Evaluator

A conceptual description of the LEE that is incorporated into the Pixel-planes system follows; a detailed description appears elsewhere.[2,3] The idea (similar to some serial multipliers[6]) is to construct a tree-structured, serial-parallel multiplier which takes as input the three bit streams A, B, and C and produces as output the value of the expression $Ax + By + C$ simultaneously at every pixel (x, y) on the screen.

To illustrate how this tree is constructed, Figure 1 shows a three-level example which will evaluate $Ax + C$ for $x = 0, 1, \ldots, 7$. The products Ax accumulate going down the tree as a series of partial products, with the appropriate multiple of A added at each level, as shown in Figure 1a. Figure 1b illustrates an efficient implementation of this idea. The three-level binary tree has at each node a one-bit adder, delay, a side bit-stream input, and a parent bit-stream input. The bit streams, least significant bit (LSB) first, are sent down the tree in the following way:

1. The left child is sent the parent stream delayed one time unit.
2. The right child is sent the sum of the parent stream and the side stream.

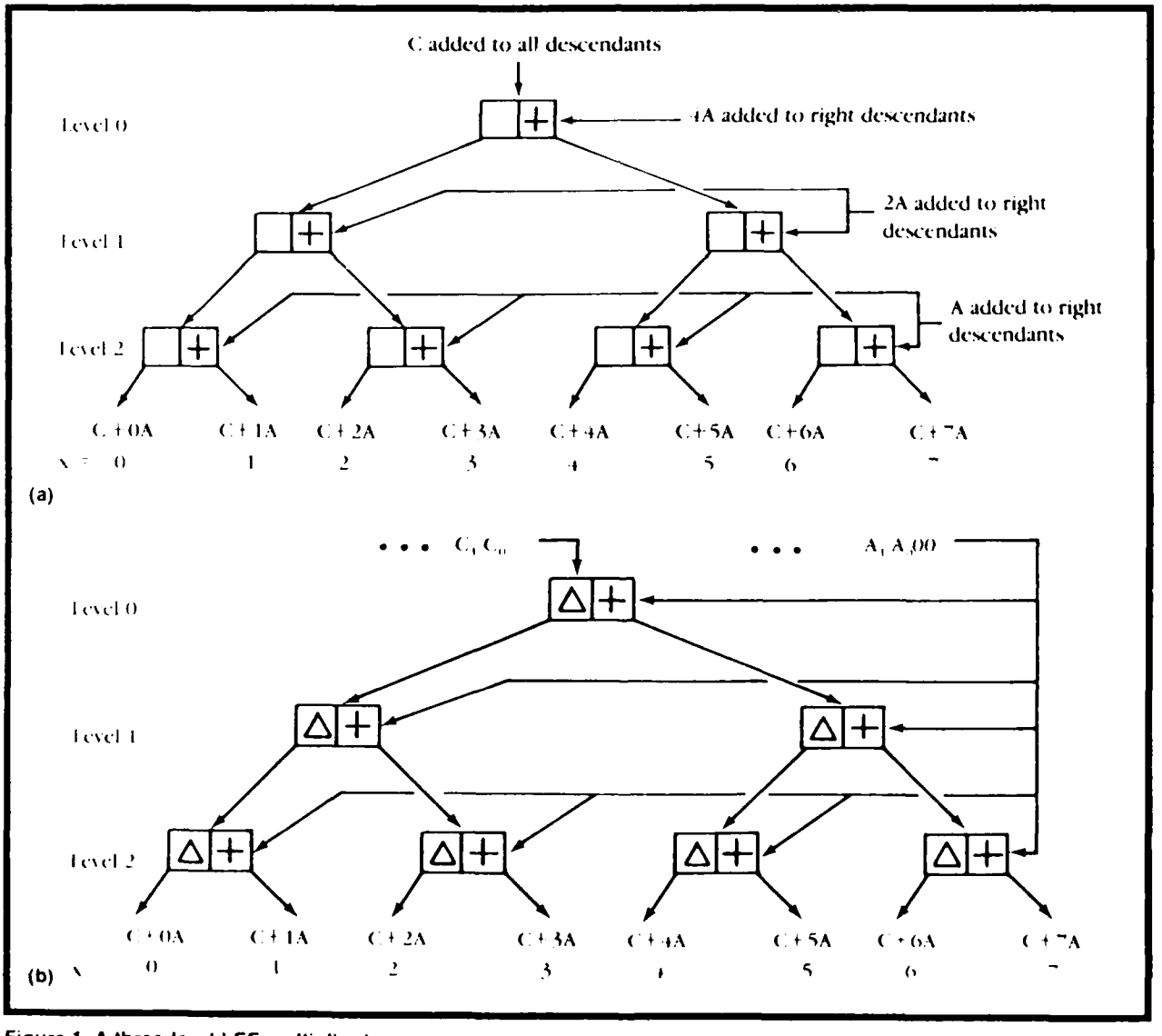


Figure 1. A three-level LEE multiplier tree.

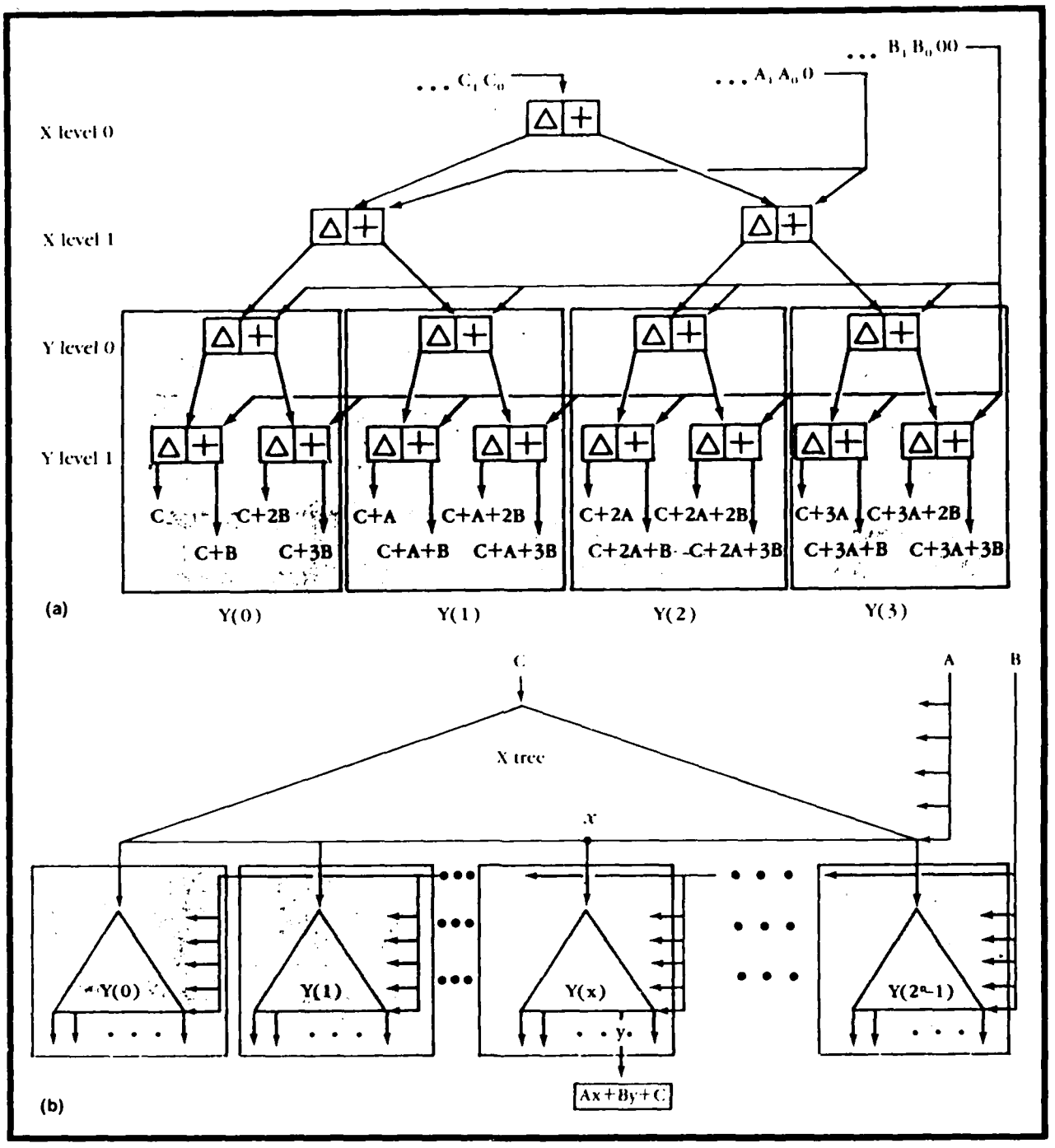


Figure 2. (a) Complete two-level XY-tree to compute Ax + By + C for x,y = 0, 1, 2, 3. (b) Schematic diagram of complete *n*-level XY-tree.

The delay to the left child is simply to keep the bit streams flowing down the left and right branches at the same rate. Note that the 0th bit of a level 0 side input reaches level 2 at the same time as the second bit from a level 2 side input. Hence, if A is the side input at level 0 (that is, the root node) it arrives at the leaf node as a $4A$, and if A is the side input at level 1, it arrives at the leaf node as a $2A$. In order to have something in the parent stream at level 2 when the LSB's of the side inputs arrive, we append two zeros in front of the LSB of each side input, and ignore the first two bits coming out of the leaf nodes.

If we want the eight bit streams emanating from the leaf nodes to be C, $C + A$, . . . , $C + 7A$, then the root input must be C, and each side input must be A with two zeros put in front of them. For example, consider $6A + C$. The binary form of 6, $(110)_2$, defines a unique path through the tree: right branch at level 0, right branch at level 1, left branch at level 0. This translates to: add A shifted twice at

level 2; add A shifted once at level 1. This, in turn, is equivalent to adding $4A + 2A + 0 = 6A$ at level 2. The root input C passes through the tree as a constant summand to each terminal bit stream.

The LEE is constructed by generalizing this design in the following way:

1. An n-level, binary adder delay X-tree is constructed, each node of which has a parent-input bit stream and a side-input bit stream. Each node itself is a one-bit adder/delay that (a) delays the bit stream from the parent and sends this parent bit stream to the left child (the parent stream to the root node is C); and (b) adds the parent bit stream to the side bit stream $2^{n-1}A$ (that is, the bit stream with $n-1$ zeros preceding the LSB of A), and sends this sum of two bit streams to the right child.

   The purpose of the delay to the left child is simply to keep the bit streams flowing down the tree at the same rate, since the add operation delays the flow to the right child.
2. There are 2^{n-1} bit streams emanating from the leaf nodes of this n-level X-tree. By writing $Ax + By + C$ in the form $By + (Ax + C)$, we see that it suffices to construct, for each X-tree, another n-level binary adder/delay tree. This tree, called the Y(x)-tree, will receive root input $Ax + C$ from the xth bit stream of the X-tree, and side input B. Because the xth bit stream has already been delayed by $n-1$ as it passed through the X-tree, we must add $2n-2$ zeros in front of the LSB of B.
3. The bit stream emanating from the yth leaf node of the Y(x)-tree represents the value of $Ax + By + C$ for the pixel (x, y). Figure 2a illustrates a complete, small X-Y tree with two levels for X and two levels for Y. Such a LEE would suffice for a trivially small memory chip for a frame buffer with two scan lines and two pixels per scan line. Figure 2b is a schematic of the general X-Y tree.

Several observations about this construction will be useful in our discussion of the quadratic version of the LEE later in this article.

Leading zeros. The zeros that precede the LSB of each bit stream as a result of its multiplication by a power of two are necessary to "initialize" the computation. That is, the zeros are needed in every parent stream when the LSB of A in the X-tree (or B in the Y-tree) arrives from the side input. The appropriate number of "early arriving" bits to each pixel are discarded. For the rest of this article, unless otherwise indicated, we will omit mention of these leading zeros. For example, we will say the side inputs to the X-tree are A rather than $2^{n-1}A$.

Node labels. The effect of adding A at a node at level k ($k = 0 \ldots n-1$) in the X-tree is that of adding $2^{n-k-1}A$ to all pixels which are right descendants of that node. Suppose each node at level k is labeled with the binary number $(b_0 b_1 \ldots b_{k-1} 100 \ldots 0)_2$ (b_0 is the most significant bit) where b_i is 1 if the node can be traced back to a right branch at level i, and is 0 otherwise. The root node ($k = 0$) receives the label $100 \ldots 0$. If $x = (b_0 b_1 \ldots b_{n-1})_2$, then the value that accumulates at location x in the X-tree is

$$\text{Root Input} + \sum_{i=0}^{n-1} 2^{n-i-1} b_i \cdot (\text{Side Input at node } b_0 b_1 \ldots b_{i-1} 10 \ldots 0) = C + \sum_{i=0}^{n-1} 2^{n-i-1} b_i A = Ax + C$$

The xy term. Although not presently implemented in the Pixel-planes system, the outputs from the X-tree could be rerouted into the side inputs of the Y(x) trees, rather than into the root node. This modified LEE would produce Bxy when B is the side input into the X-tree.

Path design. Technology constraints make it impossible to put the entire tree on a single chip. A portion of the Y(x)-tree is put on the chip together with the path from the root node of the X-tree to this portion of the Y(x)-tree, as shown in Figure 3. The path is activated on each chip by using the binary code discussed above.

There are several possible schemes for the QEE, some of which may look simpler than others from the point of view of the global layout, but the one that we chose seems to be the most promising, considering the constraints mentioned above.

The key idea in the QEE design is to send a different side input to every node in the X- and Y (x)-trees.

The Quadratic Expression Evaluator

We now illustrate the construction of an enhanced tree structure that accepts as input the six coefficients (A, B, C, D, E, F) and produces as output the expression $Ax^2 + Bxy + Cy^2 + Dx + Ey + F$ simultaneously for every pixel (x, y). The key idea in the design of the QEE is to send a different side input to every node in the X- and Y(x)-trees, rather than the same inputs as in the LEE. Recall that each node of the X-tree at level k can be labeled by the binary number $(b_0 b_1 \ldots b_{k-1} 10 \ldots 0)_2$ and that if

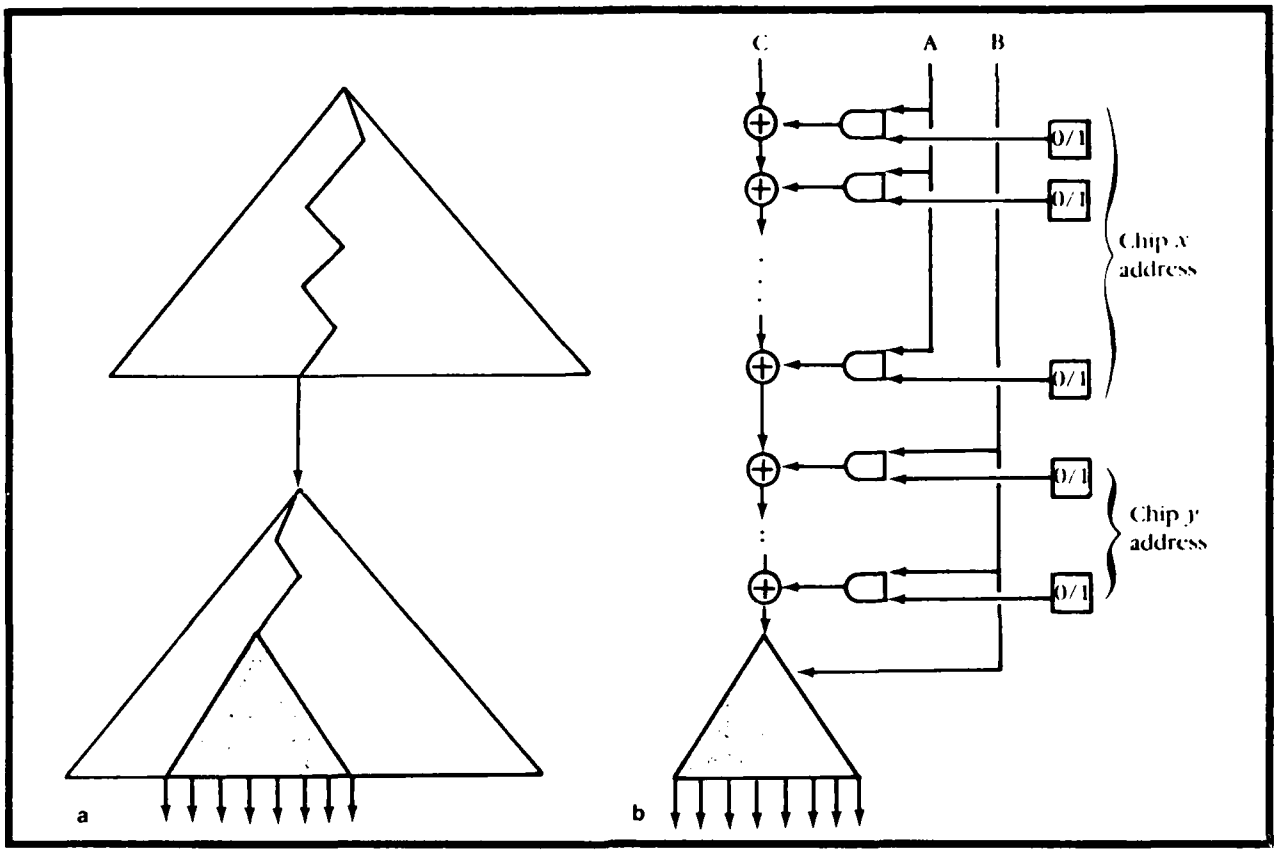


Figure 3. Path through X-tree and part of Y(x)-tree.

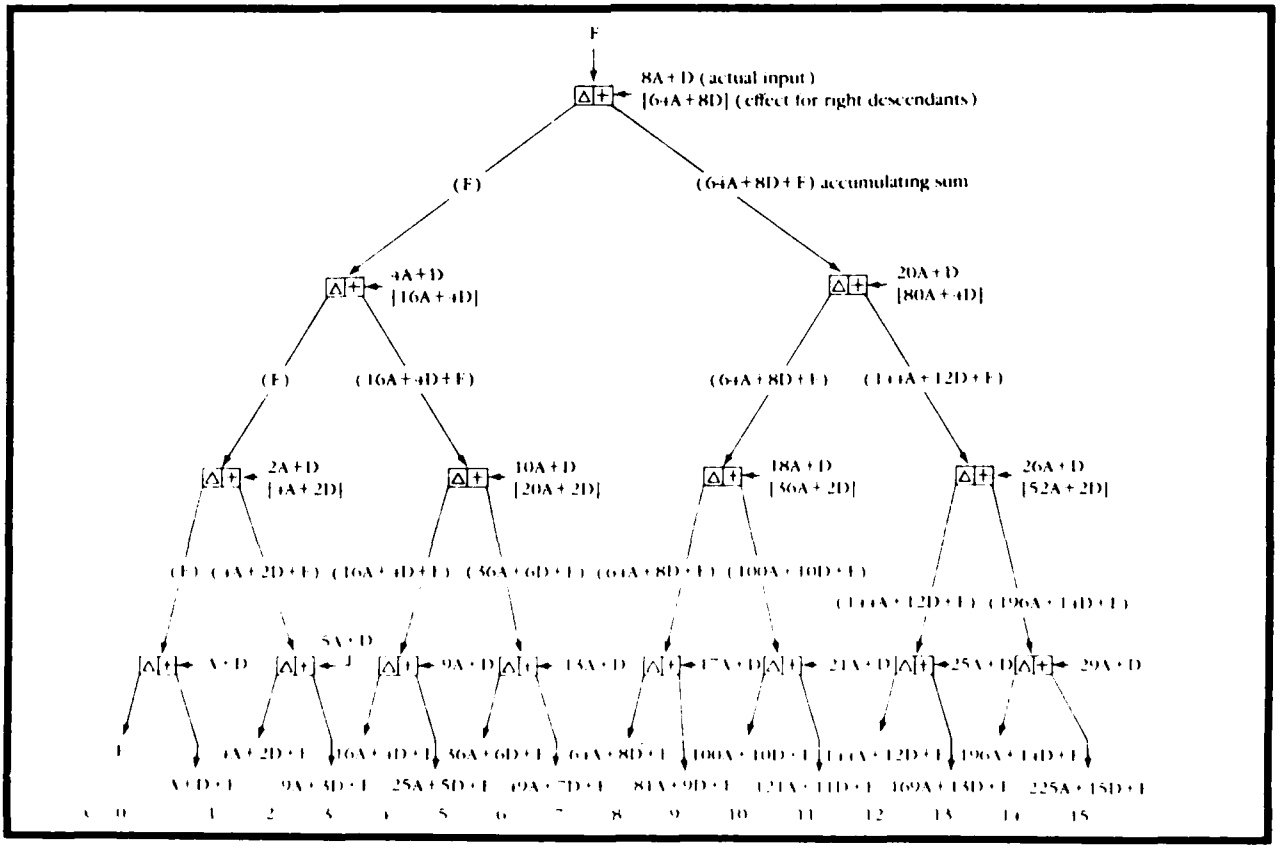


Figure 4. Side inputs for a four-level tree to produce $Ax^2 + Dx + F$ for x 0, 1, . . . 15.

$x = (b_0 b_1 \ldots b_{n-1})_2 = \sum_{k=0}^{n-1} 2^{n-k-1} b_k$

then the x bit stream is

Root Input +

$\sum_{k=0}^{n-1} 2^{n-k-1} b_k \cdot$ (Side Input at $b_0 b_1 \ldots b_{k-1}\, 10 \ldots 0$)

We modify the side inputs to the X-tree to generate the expression $Ax^2 + Dx + F$. Since we already know how to evaluate $Dx + F$ (using side input D and root input F), we will concentrate for the moment on the Ax^2 part of the expression. Suppose we write x in the binary expansion form

$$x = \sum_{k=0}^{n-1} 2^{n-k-1} b_k$$

Then

$$x^2 = \left(\sum_{k=0}^{n-1} 2^{n-k-1} b_k\right)^2 =$$

$$\sum_{k=0}^{n-1} b_k^2 2^{2(n-k-1)} + 2 \sum_{k=0}^{n-1} \sum_{j=0}^{k-1} b_k b_j 2^{n-k-1} 2^{n-j-1}$$

If we observe that $b_k^2 = b_k$ (since $b_k = 0$ or 1) and use the convention that if $k = 0$ then $\sum_{j=0}^{k-1} (anything) = 0$, then we can write Ax^2 as

$$\sum_{k=0}^{n-1} 2^{n-k-1} b_k \left(2^{n-k-1}\left(A + \sum_{j=0}^{k-1} 2^{k-j} b_j (2A)\right)\right)$$

Hence if we let the root input be F, and the side input to node $(b_0 b_1 \ldots b_{k-1}\, 10 \ldots 0)_2$ be

$$D + 2^{n-k-1}\left(A + \sum_{j=0}^{k-1} 2^{k-j} b_j (2A)\right)$$

then the x bit stream is

Root input $= \sum_{k=0}^{n-1} 2^{n-k-1} b_k$ (Side input at $b_0 b_1 \ldots b_{k-1}\, 10 \ldots 0$)

$$F + \sum_{k=0}^{n-1} 2^{n-k-1} b_k \left(D + 2^{n-k-1}\left(A + \sum_{j=0}^{k-1} 2^{k-j} b_j (2A)\right)\right)$$

$$= F + D \sum_{k=0}^{n-1} 2^{n-k-1} b_k + \sum_{k=0}^{n-1} 2^{n-k-1} b_k \left(2^{n-k-1}\left(A + \sum_{j=0}^{k-1} 2^{k-j} b_j (2A)\right)\right)$$

$$= F + Dx + Ax^2$$

See Figure 4 for a four-level example.

The problem of generating $Ax^2 + Dx + F$ has now been reduced to computing these side inputs to the X-tree. The key observation is that the summand

$$\sum_{j=0}^{k-1} 2^{k-j} b_j (2A)$$

can be evaluated by "siphoning off" the left child output of the corresponding node of another n-level tree with root input 0 and a constant side input $2A$.

Putting this all together, the side inputs into the X-tree that are necessary to evaluate the expression $Ax^2 + Dx + F$ at location x can be generated as follows:

1. A new "PX-tree" of adders identical to the X-tree is sent a root input of 0, and a constant side input of $2A$.
2. The left child output of a node at level k in the PX-tree, in addition to being sent down the PX-tree in the usual way, is (a) added to A, delayed $(n-1-k)$ time units, added to D; and (b) becomes the side input to the corresponding node of the X-tree.

Figure 4 shows an example of a four-level tree to evaluate $Ax^2 + Dx + F$, and Figure 5a illustrates the calculation that generates side inputs for the top two levels of the tree. Figure 5b is a schematic of the general node-to-node linking, referred to as a *Quadratic Linked X-tree* with inputs (A, D, F).

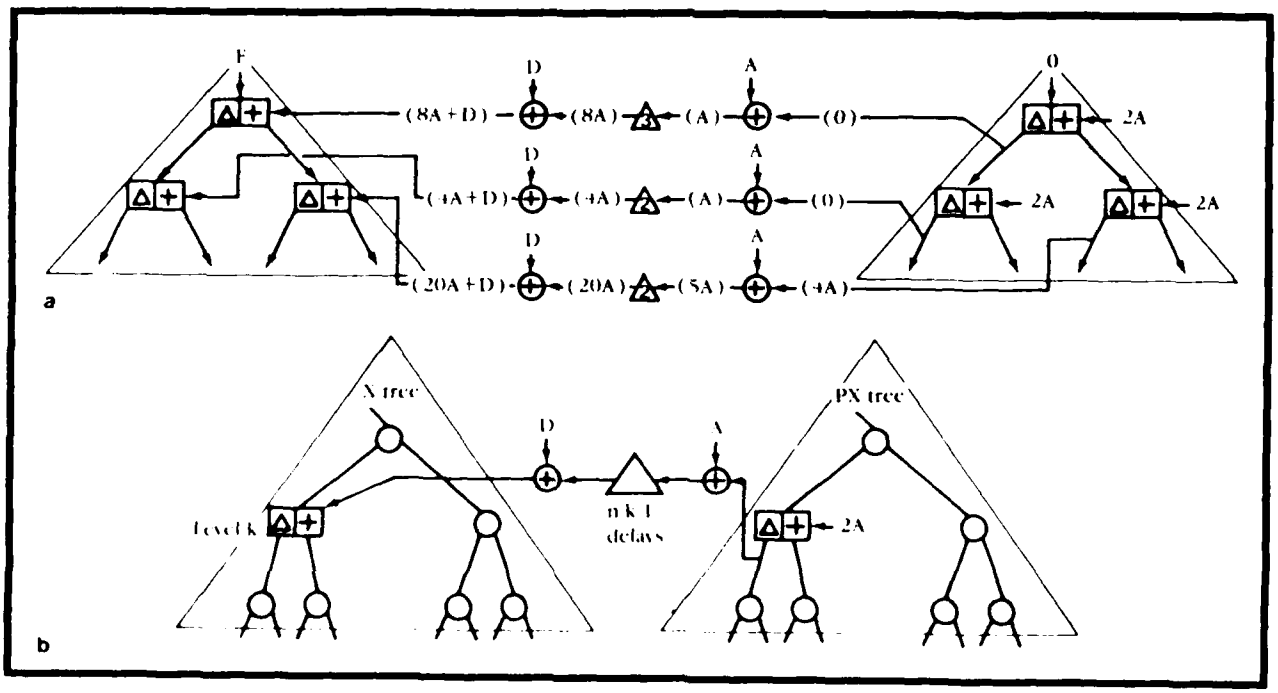


Figure 5. (a) Creating inputs for the top two levels of the tree in Figure 4. (b) PX-to-X node connection at level k.

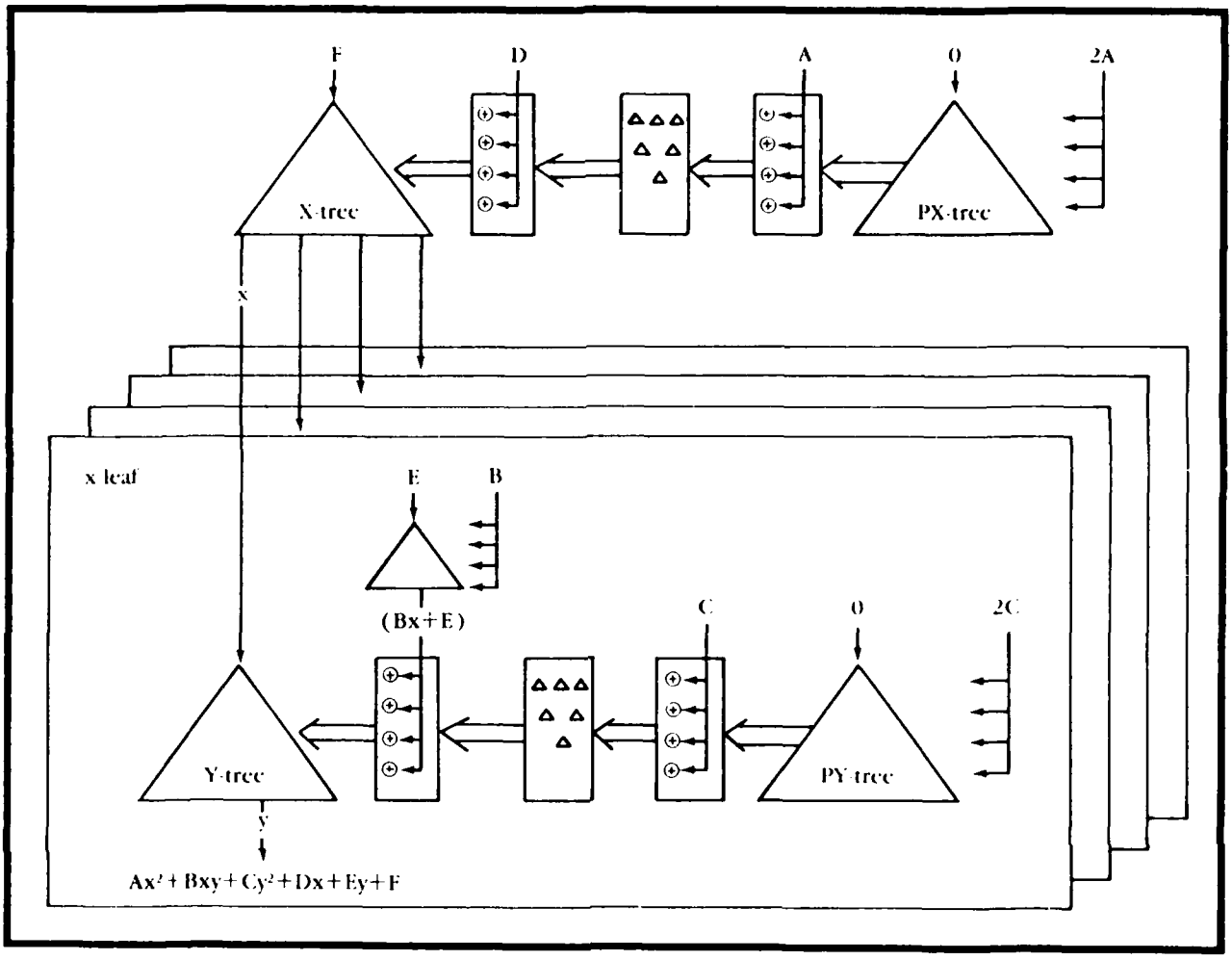


Figure 6. Schematic layout of the QEE.

The quadratic expression in two variables is an extension of this scheme. Suppose we write the expression $Q(x, y) = Ax^2 + Bxy + Cy^2 + Dx + Ey + F$ in the form $Cy^2 + (Bx + E)y + (Ax^2 + Dx + F)$. Then, to evaluate $Q(x, y)$ it suffices to construct a Quadratic Linked Y(x)-tree with inputs $(C, Bx + E, Ax^2 + Dx + F)$. The last two inputs are generated via a Linear X-tree and a Quadratic Linked X-tree, respectively. Figure 6 shows the complete schematic of the QEE.

Since the entire QEE for all pixels on the screen cannot be contained on a single chip, we proceed in a manner analogous to the path-to-subtree scheme of the LEE. On each chip, we put as much of the linked PY(x)- and Y(x)-trees as are needed for the pixels on this chip. Three identical, complete X paths and two identical, partial Y paths are constructed, and linked appropriately to replicate that portion of the complete QEE relevant to the pixels on this chip. An illustration of this is given in Figure 7. Figure 8 illustrates the PY(x)- and Y(x)-subtrees that would be implemented on a small 16-pixel chip.

Implementing a system with a QEE

We have just begun to plan the implementation of Pixel-powers. It appears so far to be a straightforward expansion of the Pixel planes implementation. A brief look at that implementation may help explain the one being planned.

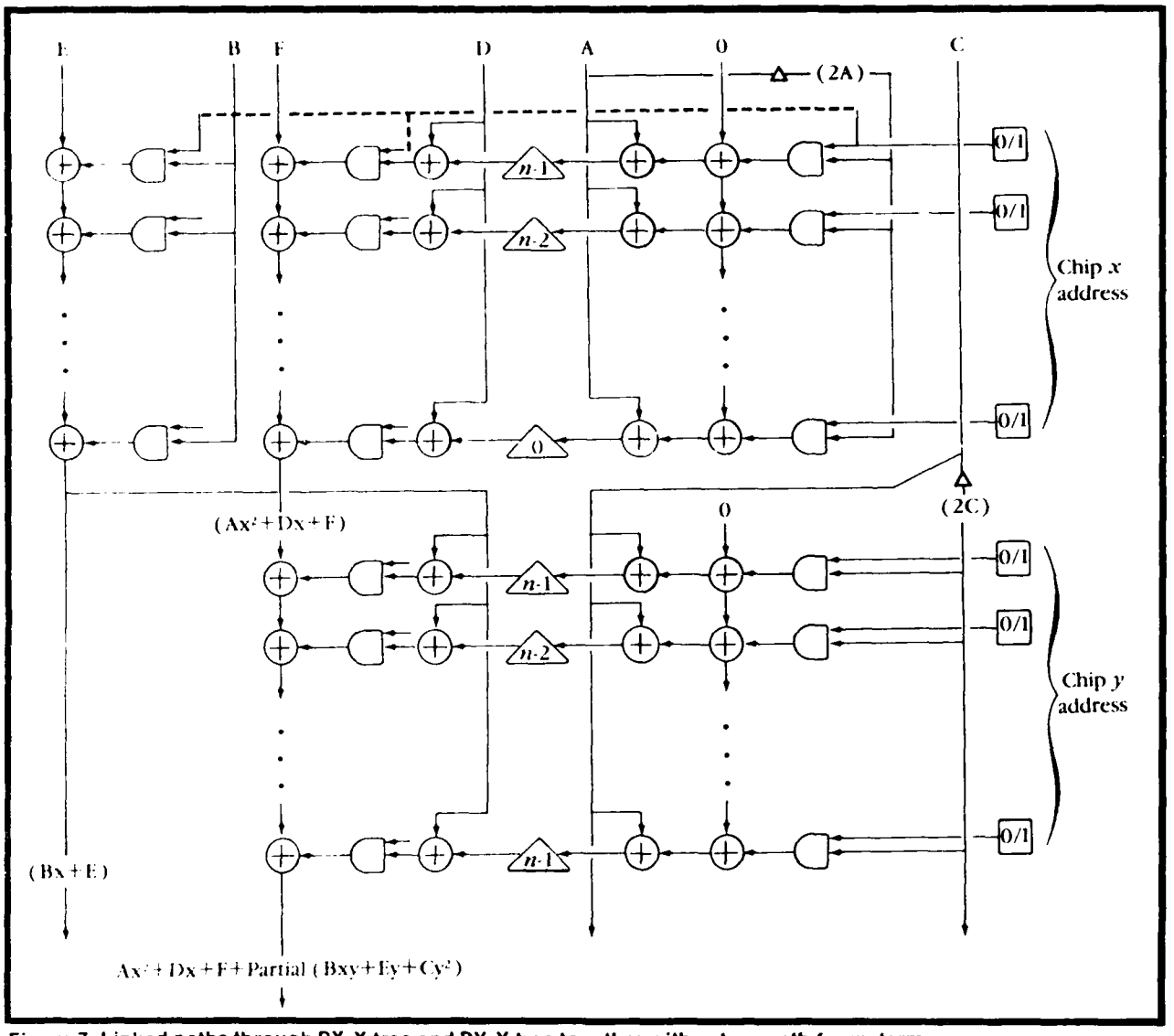


Figure 7. Linked paths through PX-X tree and PY-Y tree together with extra x path for xy term.

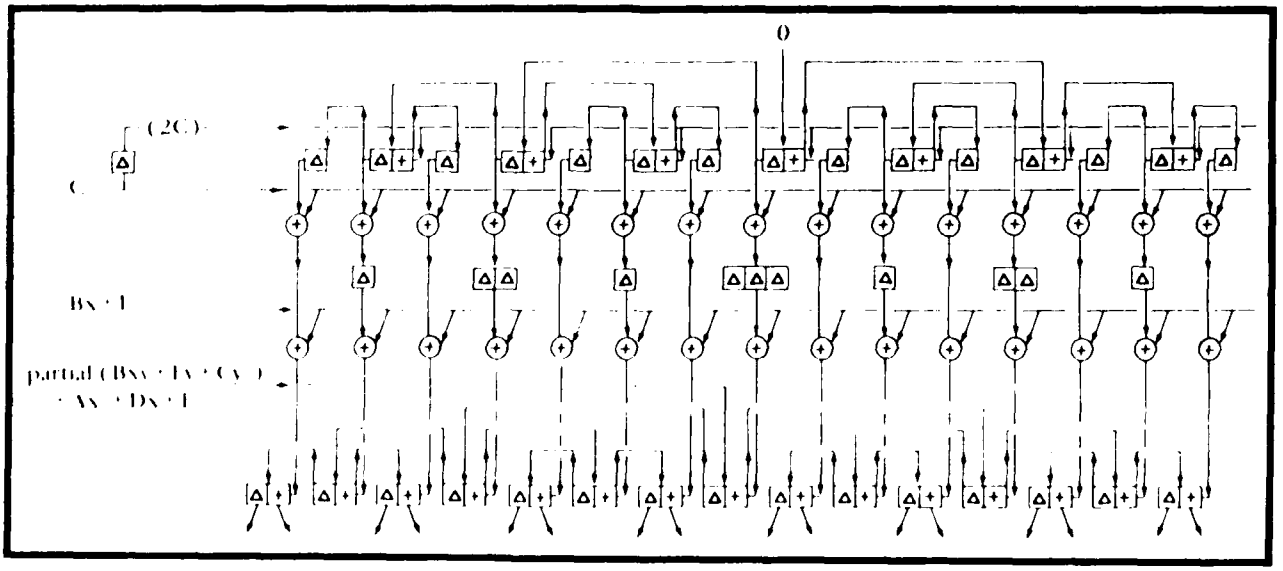


Figure 8. Chip organization of linked subtrees.

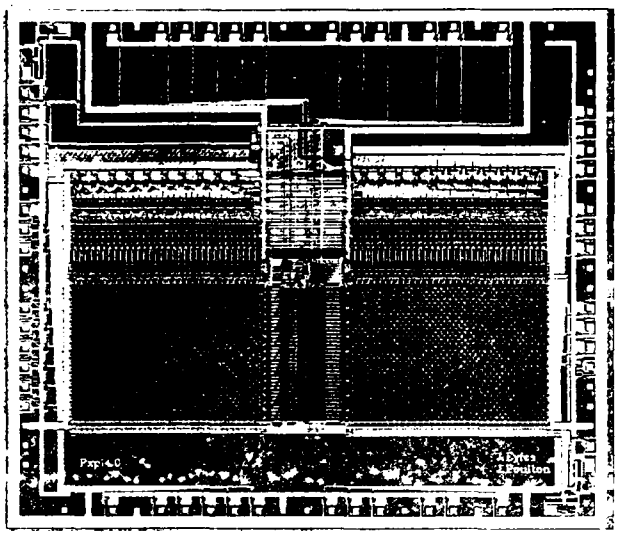


Figure 9. Micrograph of Pixel-planes4.0 memory chip (Melgar Photographers).

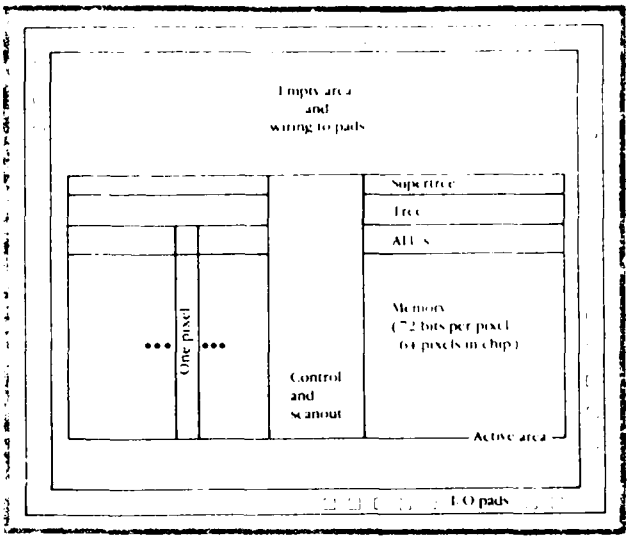


Figure 10. Pixel-planes4.0 general floor plan.

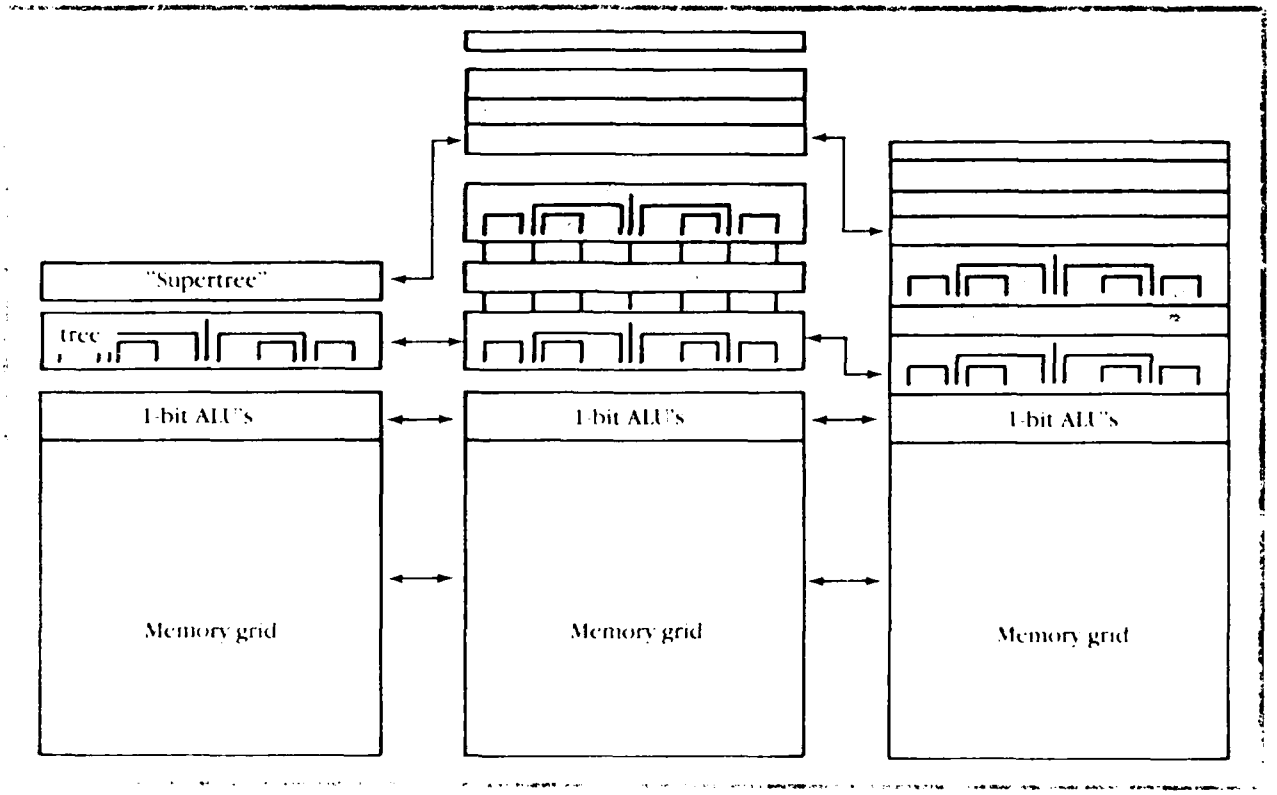
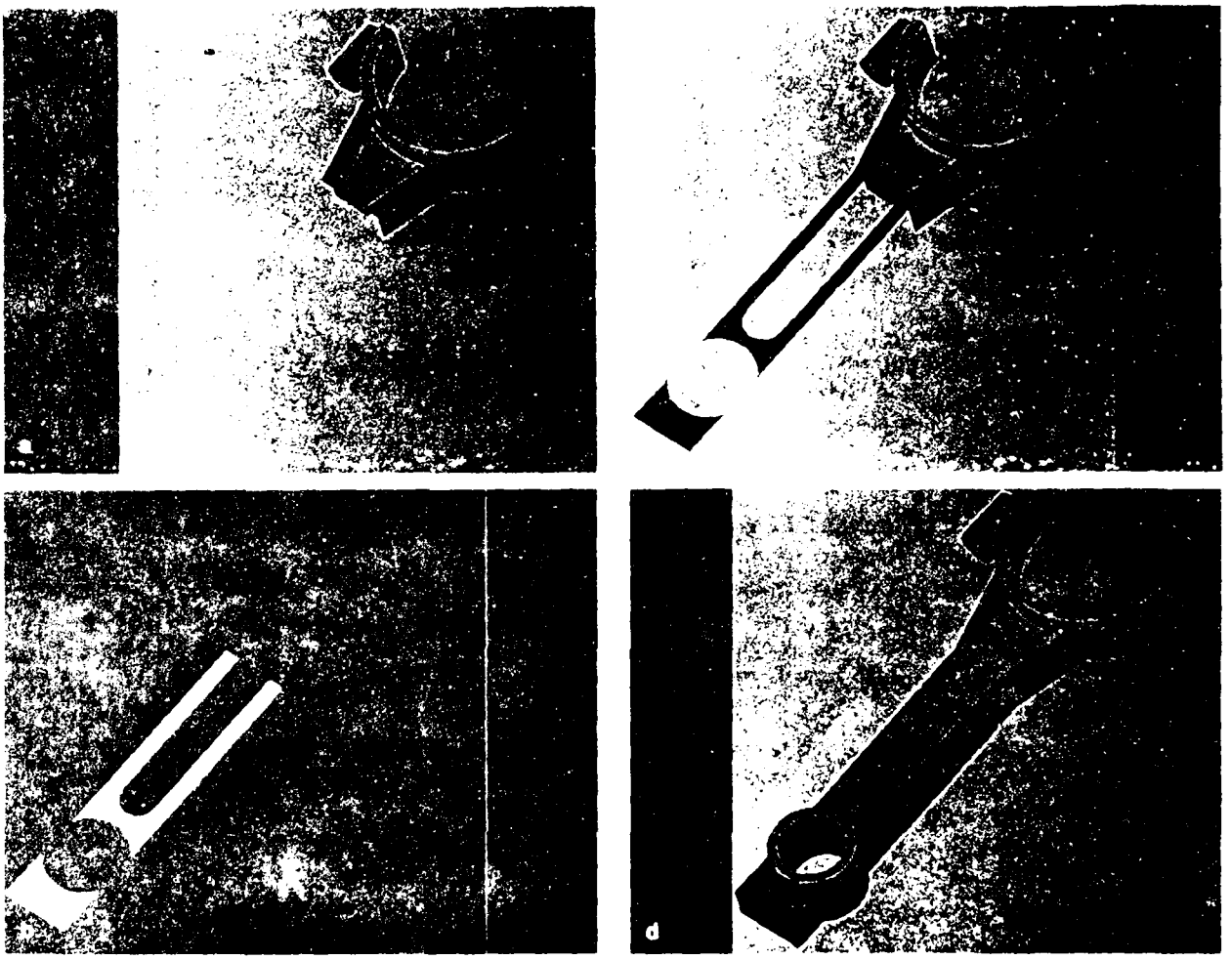


Figure 11. Pixel-planes to Pixel-powers organization (common central wiring channel not shown).

Figure 9 is a micrograph of our fourth-generation Pixel-planes4.0 memory chip, and Figure 10 is a general floor plan of this chip. (We actually have a newer, larger chip that is now working, Pixel-planes4.1, but since that chip is fundamentally two Pixel planes4.0 chips, it is just as valid, but simpler, to use Pixel planes4.0 as the basis of comparison.) To implement a QEE within such a system, we put a QEE in place of the LEE.

The LEE on a chip consists of two modules: the portion of the complete binary "multiplier tree" for the pixels on this chip, and the extra tree path ("supertree") to the root of the global tree. Figure 11 illustrates the transformation



12. All images are produced by a functional simu- ... final image after processing of 190 opcodes ... of the test sets (16 opcodes do not need coef- ... opcodes specify the area and depth of the ... that are subtracted by other ... pixels are copied into the z buffer and those still visible are shaded—total of opcodes is now 222; (*d*) completed simulation with a final total of 342 opcodes with 260 coefficient sets. Time to generate this image in a 10-MHz chip is estimated to be 900 microseconds.

CSG-defined object in Pixel-powers – a connecting rod of an internal combustion engine. The object has 17 primitives, of which 14 are evident in this image. Seventy coefficient sets (*A, B, C, D, E, F*) were required to generate the image. We estimate that the object could be generated by Pixel-powers memory chips in less than 900 μs. The image-generation process itself is described in an upcoming report.

Conclusions

The generalization of the LEE tree to a QEE gives vastly increased power to our logic-enhanced memory chips. Since the number of primitives in a curved surface model of an object is typically much less than a polygonal model of it, the effect of the extra power in the memory chip is even more dramatic. The challenge now is to develop algorithms and systems to convert efficiently the geometrically transformed representation into a form suitable for a frame-buffer system composed of these chips. ■

Acknowledgments

We thank Jeff Hultquist for developing the functional simulator that generated the images in Figure 12, and John Eyles for developing a detailed simulator of the QEE. We also thank both Hultquist and Eyles, and John Poulton for valuable discussions and suggestions. We thank Paul Deitz and Paul Stay of the U.S. Army Ballistic Research Laboratory for the CSG data of the connecting rod.

This research is supported in part by the Defense Advanced Research Projects Agency, monitored by the U.S. Army Research Office, Research Triangle Park, North Carolina, under contract number DAAG29-83-K-0148, and the National Science Foundation grant number ECS-8300970.

References

1. Henry Fuchs and John Poulton, "Pixel-planes: A VLSI-Oriented System for a Raster Graphics Engine," *VLSI Design* (formerly *Lambda*), Vol. 2, No. 3, 3rd quarter 1981, pp.20-28.
2. Henry Fuchs, Jack Goldfeather, Jeff P. Hultquist, Susan Spach, John D. Austin, Frederick P. Brooks, Jr., John G. Eyles, and John Poulton, "Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-Planes," *Computer Graphics*, Vol. 19, No. 3, July 1985 (*Proc. SIGGRAPH 85*).
3. John Poulton, Henry Fuchs, John D. Austin, John G. Eyles, Justin Heinecke, Cheng-Hong Hsieh, Jack Goldfeather, Jeff P. Hultquist, and Susan Spach, "Implemention of a Full Scale Pixel-planes System," in *Proc. 1985 Chapel Hill Conference on VLSI*, H. Fuchs, ed., Computer Science Press, Rockville, Md.
4. Gershon Kedem and John L. Ellis, "Computer Structures for Curve-Solid Classification in Geometric Modeling," technical report TR84-37, Microelectronic Center of North Carolina, Research Triangle Park, N.C., Sept. 1984.
5. A.A.G. Requicha, "Representation for Rigid Objects: Theory, Methods, and Systems," *ACM Computing Surveys*, Vol. 12, No. 4, Dec. 1980, pp.437-464.
6. Richard F. Lyon, "Two's Complement Pipeline Multipliers," *IEEE Trans. Communications*, Vol. COM-24, April 1976, pp.481-425.

Jack Goldfeather is an associate professor of mathematics at Carleton College in Northfield, Minnesota, where he teaches a variety of undergraduate mathematics and computer science courses. His primary research in mathematics is in the area of algebraic topology, especially the algebraic properties of mappings between infinite, dimensional topological spaces. During a 1984-1985 sabbatical at UNC at Chapel Hill, he was a mathematics consultant to the Pixel-planes research group.

Goldfeather received a BA in mathematics from Rutgers University in 1969, and an MS and PhD in mathematics from Purdue University in 1971 and 1975, respectively. He taught at the University of Wisconsin at Milwaukee from 1975 to 1977 before joining the Carleton faculty.

Henry Fuchs is a professor of computer science at the University of North Carolina at Chapel Hill, where he has been teaching graduate courses in computer graphics and VLSI design, and directs the research of PhD students and research associates in graphics algorithms and VLSI architectures.

Fuchs is the principal investigator of research projects funded by DARPA, NIH, and NSF. He consults for a variety of industrial organizations. He is an associate editor of *ACM Transactions on Graphics* and was chairman of the 1985 Chapel Hill Conference on VLSI. He received a BA from the University of California at Santa Cruz in 1970, and a PhD from the University of Utah in 1975.

Jack Goldfeather can be contacted at Carleton College, Mathematics Department, One N. College St., Northfield, MN 55057; Henry Fuchs can be contacted at the University of North Carolina, Department of Computer Science, New West Hall 035A, Chapel Hill, NC 27512.

END

DTIC

7-86